



TP : Créer votre première application Android Compose

1. Avant de commencer

Si ce n'est pas déjà fait, installez [Android Studio](#) sur votre ordinateur. Vérifiez que votre ordinateur respecte la configuration système requise pour exécuter Android Studio (située en bas de la page de téléchargement).

Dans ce TP vous allez créer votre première application Android avec un modèle de projet fourni par Android Studio. Vous allez utiliser **Kotlin** et **Jetpack Compose** pour personnaliser votre application.

Notez qu'Android Studio est mis à jour fréquemment et que parfois l'interface utilisateur change. Il est donc normal que votre version d'Android Studio diffère légèrement des captures d'écran présentées dans ce TP.

Conditions préalables

- Connaissances de base en Kotlin

Ce dont vous avez besoin

- La dernière version d'Android Studio

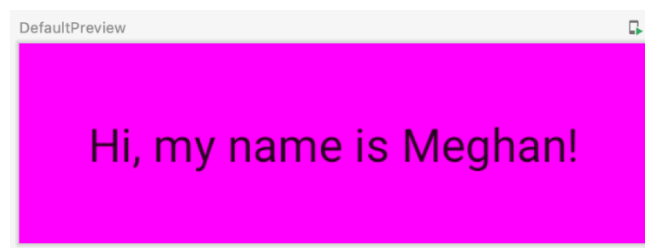
Points abordés

- Comment créer une application Jetpack Compose avec Android Studio
- Comment exécuter des applications avec l'outil de prévisualisation d'Android Studio
- Comment mettre à jour du texte avec Kotlin
- Comment mettre à jour une interface utilisateur (UI) avec Jetpack Compose
- Comment afficher un aperçu de votre application dans Jetpack Compose

Objectifs de TP

- Créer une application pour vous présenter de manière personnalisée.

Voici à quoi ressemblera l'application lorsque vous aurez terminé ce TP (sauf qu'elle sera personnalisée avec votre nom) :



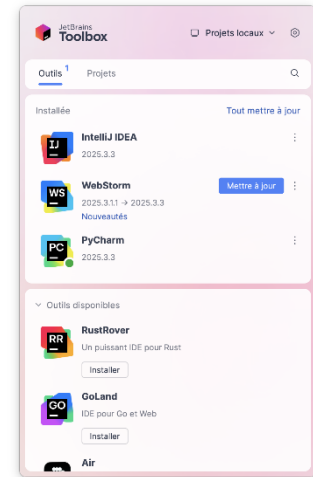
2. Installation Android Studio

Pour installer Android Studio via **JetBrains Toolbox**, suivez ces étapes simples. **JetBrains Toolbox** est un outil qui facilite la gestion et l'installation des différents IDE de JetBrains et pas seulement, y compris Android Studio. Voici comment procéder :

1. **Téléchargez et installez JetBrains Toolbox :**



- Rendez-vous sur le site officiel de JetBrains à l'adresse <https://www.jetbrains.com/toolbox-app/>.
- Téléchargez la version de JetBrains Toolbox compatible avec votre système d'exploitation (**Windows, macOS ou Linux**).
- Exécutez le fichier téléchargé et suivez les instructions à l'écran pour installer JetBrains Toolbox.



2. Ouvrez JetBrains Toolbox :

- Une fois l'installation terminée, ouvrez JetBrains Toolbox. Vous devriez voir une interface listant les différents outils et IDE disponibles à l'installation.

3. Trouvez Android Studio :

- Dans la barre de recherche ou dans la liste des produits disponibles, cherchez "**Android Studio**". Android Studio devrait apparaître dans les résultats.

4. Installez Android Studio :

- À côté d'Android Studio, vous verrez un bouton d'installation. Cliquez dessus pour commencer l'installation.
- Vous pouvez avoir la possibilité de choisir le chemin d'installation ainsi que d'autres options d'installation. Configurez-les selon vos préférences.
- L'outil téléchargera et installera Android Studio. Ce processus peut prendre un **certain temps** en fonction de votre connexion internet.

5. Lancez Android Studio :

- Une fois l'installation terminée, vous pouvez lancer Android Studio directement depuis JetBrains Toolbox en cliquant sur l'icône, le bouton "Ouvrir" ou "Lancer" à côté d'Android Studio dans votre liste d'applications JetBrains.

6. Configuration initiale :

- Lors du premier lancement d'Android Studio, vous serez guidé à travers le processus de configuration initiale, y compris le téléchargement des composants **SDK Android** nécessaires, la configuration de l'environnement de développement, etc.

7. Mise à jour d'Android Studio :

- JetBrains Toolbox vérifie automatiquement les mises à jour pour Android Studio et vous permet de les installer facilement. Vous serez notifié lorsqu'une mise à jour est disponible, et vous pourrez la mettre à jour **directement depuis l'interface de Toolbox**.

En utilisant JetBrains Toolbox, vous bénéficiez d'une gestion centralisée de vos outils de développement, facilitant l'installation, la mise à jour et le lancement d'Android Studio et d'autres IDE JetBrains ainsi que vos projets récents disponibles directement dans l'onglet « Projets ».

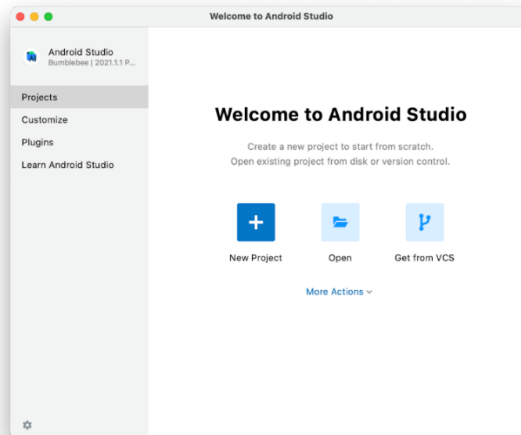


3. Créer un projet à l'aide du modèle

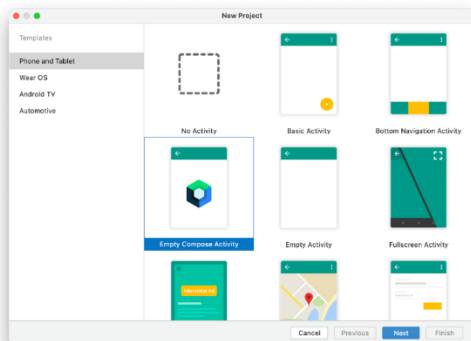
Dans ce TP, vous allez créer une application Android avec le modèle de projet **Activité Compose vide** fourni par Android Studio.

Pour créer un projet dans Android Studio :

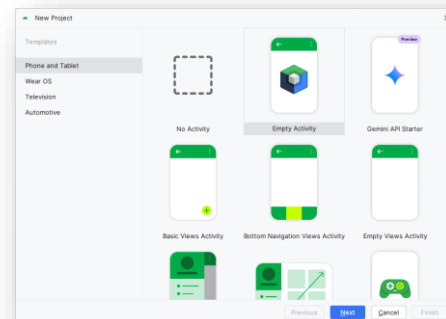
1. Lancez Android Studio .
2. Dans la boîte de dialogue **Bienvenue dans Android Studio**, cliquez sur **Nouveau projet**.



La fenêtre **Nouveau projet** s'ouvre et affiche la liste des modèles fournis par Android Studio.



Ancienne version d'Android Studio



Nouvelle version d'Android Studio

Dans Android Studio, un modèle de projet est un projet Android qui fournit le plan pour un certain type d'application. Les modèles créent la structure du projet et les fichiers dont Android Studio a besoin pour créer votre projet. Le code de démarrage mis à votre disposition pour une mise en route rapide dépend du modèle que vous choisissez.

3. Assurez-vous que l'onglet **Téléphone et tablette** est sélectionné.
4. Cliquez sur le modèle **Empty Compose Activity** (ou bien **Empty Activity** pour les versions récentes) afin de le sélectionner comme modèle pour votre projet. Ce modèle "**Activité Compose vide**" sert à créer un projet simple que vous pouvez utiliser pour compiler une application **Compose**. Il comprend un seul écran et affiche le texte "**Hello Android!**".



5. Cliquez sur **Suivant**. La boîte de dialogue **Nouveau projet** s'ouvre. Elle contient des champs permettant de configurer votre projet.
6. Configurez votre projet comme suit :

Le champ **Nom** permet de saisir le nom de votre projet. Pour ce TP, saisissez "**GreetingCard**".

Ne modifiez pas le champ **Nom du package**. C'est de cette façon que vos fichiers seront organisés dans la structure de fichiers. Dans le cas présent, le nom du package sera `com.example.greetingcard`.

Laissez le champ **Emplacement d'enregistrement** tel quel. C'est là que sont enregistrés tous les fichiers associés à votre projet. Prenez note de cet emplacement sur votre ordinateur pour pouvoir retrouver vos fichiers.

Kotlin est déjà sélectionné dans le champ **Langage**. Il s'agit du langage de programmation que vous souhaitez utiliser pour votre projet. Vous ne pouvez pas modifier ce champ, car Compose n'est compatible qu'avec Kotlin.

New Project

Empty Activity

Create a new empty activity with Jetpack Compose

Name: GreetingCard

Package name: com.example.greetingcard

Save location: C:\Users\Y\AndroidStudioProjects\JC\GreetingCard

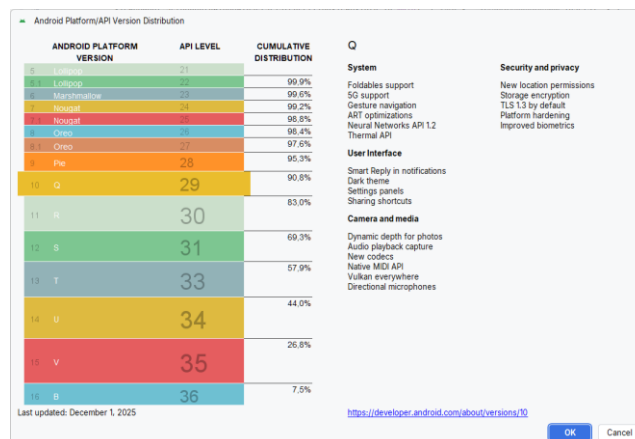
Minimum SDK: API 29 (*Q*; Android 10.0)

Your app will run on approximately 90.8% of devices. [Help me choose](#)

Build configuration language: Kotlin DSL (build.gradle.kts) [Recommended]

Previous Next Cancel Finish

Minimum SDK - Vous avez ici la possibilité de choisir le SDK minimum avec lequel votre application sera compatible, le niveau **minimum recommandé est API 21** ou supérieur. L'interface vous indique que le SDK proposé par défaut (**API 29 : Android 10.0 "Q" – pour « Quince Tart », une tarte aux coings**) en mai 2026, sera compatible avec environ 90,8% des périphériques Android en circulation. Conservez donc cette valeur.



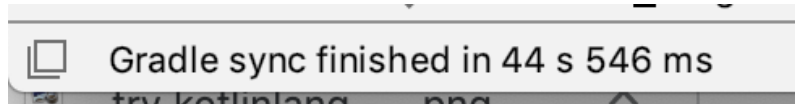
Derrière le lien « **Help me choose** »



- Appuyez sur **Terminer (Finish)**. Cette opération peut prendre un certain temps. Pendant la configuration d'Android Studio, une barre de progression et un message indiquent si la configuration du projet est toujours en cours. Cela peut se présenter comme suit :



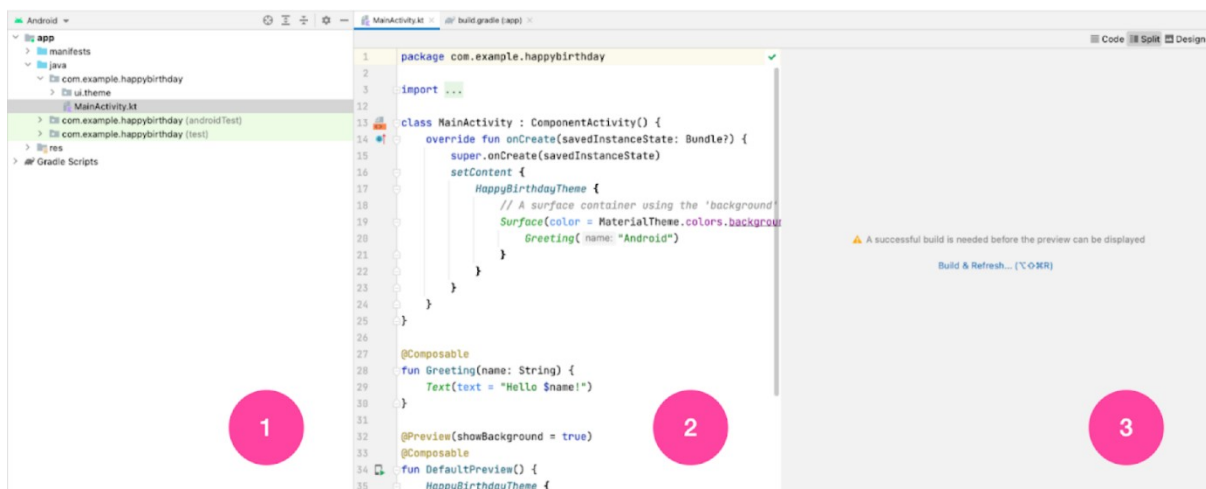
Un message semblable à celui-ci vous informe lorsque la configuration du projet est terminée.



- Il se peut qu'un volet **Nouveautés** s'affiche pour vous présenter les nouvelles fonctionnalités d'Android Studio. Fermez-le pour l'instant.
- Cliquez sur **Diviser (Split)** en haut à droite d'Android Studio pour afficher simultanément le code et la conception (**Design**). Vous pouvez également cliquer sur **Code** pour n'afficher que le code ou sur **Design** pour n'afficher que la conception.

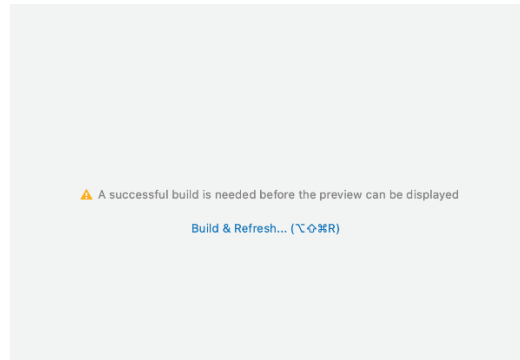


Lorsque vous appuyez sur **Diviser**, l'écran se divise en trois zones :

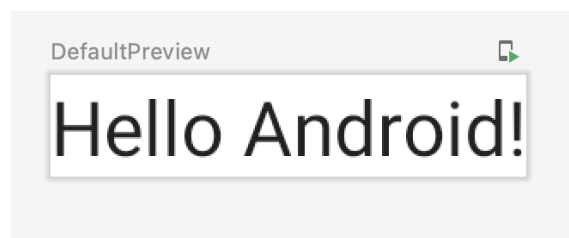


- La vue **Projet** (1) affiche les fichiers et les dossiers de votre projet.
- La vue **Code** (2) est la zone dans laquelle vous pouvez modifier le code.
- La vue **Design** (3) vous permet d'afficher un aperçu de votre application.

La vue **Design** (Conception) affiche un volet vide contenant le texte suivant :



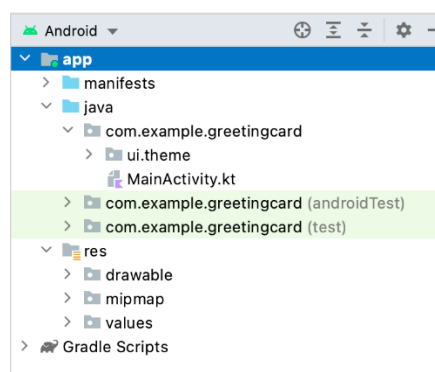
10. Cliquez sur **Compiler et actualiser (Build & Refresh)**. La compilation peut prendre un certain temps. Une fois qu'elle est terminée, l'aperçu affiche **Hello Android!** dans la zone de texte. L'activité Compose vide contient tout le code nécessaire pour créer cette application.



4. Rechercher des fichiers de projet

Dans cette section, vous allez poursuivre l'exploration d'Android Studio en vous familiarisant avec la structure de fichiers.

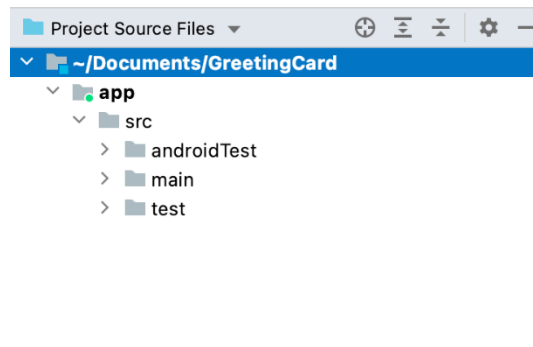
1. Dans Android Studio, examinez l'onglet **Projet**. L'onglet **Projet** affiche les fichiers et dossiers de votre projet. Lorsque vous avez configuré votre projet, le nom du package était **com.example.greetingcard**. Vous pouvez voir ce package ici même, dans l'onglet **Projet**. Un package est en fait un dossier dans lequel se trouve le code. Android Studio organise le projet dans une structure de répertoires composée de packages.
2. Si nécessaire, sélectionnez **Android** dans le menu déroulant de l'onglet **Projet**.



Il s'agit de l'affichage et de l'organisation standards des fichiers que vous utilisez. Cela s'avère utile lorsque vous écrivez du code pour votre projet, dans la mesure où vous pouvez facilement accéder aux fichiers que vous utiliserez dans votre application. Cependant, si vous consultez les fichiers dans un explorateur de fichiers, tel que le Finder ou l'Explorateur Windows, vous remarquerez que la hiérarchie des fichiers est très différente.



3. Sélectionnez **Fichiers sources du projet** dans le menu déroulant. Vous pouvez maintenant parcourir les fichiers de la même manière que dans n'importe quel autre explorateur de fichiers.



4. Sélectionnez à nouveau **Android** pour revenir à la vue précédente. Si votre structure de fichiers vous paraît étrange, assurez-vous que vous êtes toujours dans la vue Android.

5. Modifier le texte

Maintenant que vous vous êtes "familiarisé" 😊 avec Android Studio, il est temps de créer votre carte de présentation.

Examinez la vue **Code** du fichier **MainActivity.kt**. Notez que ce code contient des fonctions générées automatiquement, et en particulier **onCreate()** et **setContent()**.

Remarque : Gardez à l'esprit qu'une fonction est un segment d'un programme qui exécute une tâche spécifique.

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            GreetingCardTheme {
                Scaffold(modifier = Modifier.fillMaxSize()) { innerPadding ->
                    Greeting(
                        name = "Android",
                        modifier = Modifier.padding(innerPadding)
                    )
                }
            }
        }
    }
}
```

La fonction **onCreate()** est le point d'entrée de cette application et elle appelle d'autres fonctions pour créer l'interface utilisateur. Dans les programmes Kotlin, la fonction **main()** est le point précis où démarre le compilateur Kotlin dans votre code. Dans les applications Android, c'est la fonction **onCreate()** qui remplit ce rôle.

La fonction **setContent()** à l'intérieur de la fonction **onCreate()** permet de définir votre mise en page au moyen de fonctions modulaires. Toutes les fonctions annotées avec **@Composable** peuvent être appelées à partir de la fonction **setContent()** ou d'autres fonctions modulaires. L'annotation indique au compilateur Kotlin que cette fonction est utilisée par Jetpack Compose pour générer l'interface utilisateur.

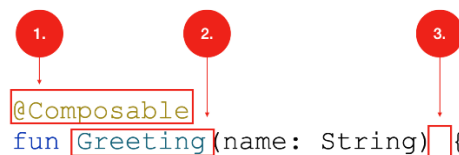


Remarque : Le compilateur utilise le code Kotlin que vous avez rédigé, l'examine ligne par ligne et le traduit en un texte compréhensible par l'ordinateur. Ce processus est appelé compilation du code. Vous le saviez déjà, j'imagine 😊.

Examinez ensuite la fonction `Greeting()`. `Greeting()` est une fonction modulable ; remarquez l'annotation `@Composable` indiquée au-dessus. Une fonction modulable reçoit des entrées et génère ce qui s'affiche à l'écran.

```
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
    Text(
        text = "Hello $name!",
        modifier = modifier
    )
}
```

Vous savez déjà ce que sont les fonctions. Cependant, il existe quelques différences avec les fonctions modulables.



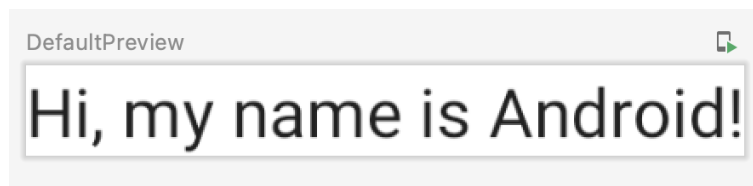
- La première lettre d'un nom de fonction `@Composable` est mise en majuscule (1).
- L'annotation `@Composable` est ajoutée avant la fonction (2).
- Les fonctions `@Composable` ne peuvent rien renvoyer (3).

Actuellement, la fonction `Greeting()` accepte un nom et affiche le message « Hello » à la personne dont le nom a été indiqué.

1. Modifiez la fonction `Greeting()` pour vous présenter au lieu de simplement dire "Hello".

```
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
    Text(
        text = "Hi, my name is $name!",
        modifier = modifier
    )
}
```

2. Recréez l'aperçu **DefaultPreview** en appuyant sur le bouton « Build & Refresh »  en haut du volet de conception (Design) :



Félicitations ! Vous avez modifié le texte, mais il vous présente sous le nom d'Android, qui n'est certainement pas votre véritable identité. Vous allez maintenant le personnaliser pour vous présenter avec votre vrai nom.



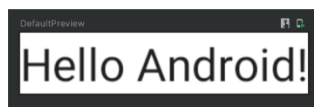
`DefaultPreview()` (ou `GreetingPreview` dans notre exemple) est une fonctionnalité particulièrement utile qui vous permet de voir à quoi ressemble votre application sans avoir à la compiler entièrement. Pour qu'il s'agisse d'une fonction d'aperçu, vous devez ajouter une annotation `@Preview`.

```
@Preview(showBackground = true)
@Composable
fun GreetingPreview() {
    GreetingCardTheme {
        Greeting("Android")
    }
}
```

Comme vous pouvez le constater, l'annotation `@Preview` accepte un paramètre appelé `showBackground`. Si `showBackground` est défini sur `true`, un arrière-plan est ajouté à l'aperçu de votre application.

Par défaut, Android Studio utilise un thème clair pour l'éditeur. Il peut donc être difficile de voir la différence entre `showBackground = true` et `showBackground = false`. Vous pouvez toutefois voir la différence ci-dessous avec un **thème sombre**. Notez que l'arrière-plan blanc de l'image est défini sur `"true"`.

showBackground = true

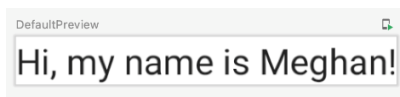


showBackground = false



3. Mettez à jour la fonction `GreetingPreview()` avec votre nom. Ensuite, recréez (**Ctrl+Maj+F5**) et vérifiez votre carte de présentation personnalisée.

```
@Preview(showBackground = true)
@Composable
fun GreetingPreview() {
    GreetingCardTheme {
        Greeting("Meghan")
    }
}
```



6. Modifier la couleur d'arrière-plan

Le texte de présentation est, il faut le reconnaître, assez terne. Dans cette section, vous allez apprendre à modifier la couleur de l'arrière-plan.

Pour définir une couleur d'arrière-plan différente pour votre présentation, vous devez placer un élément `Surface` autour du texte. **Surface** est un conteneur qui représente une section de l'interface utilisateur dans laquelle vous pouvez modifier l'apparence (la bordure ou la couleur de l'arrière-plan, par exemple).



1. Pour placer un élément **Surface** autour du texte, sélectionnez la ligne de texte, appuyez sur (**Alt+Enter** sous Windows ou **Option+Enter** sous Mac), puis sélectionnez **Entourer avec le widget** (**Surround with Widget**).

```
@Composable
fun Greeting(name: String) {
    Text(text = "Hi, my name is $name!")
}
```

2. Sélectionnez **Entourer avec le conteneur**.

```
@Composable
fun Greeting(name: String) {
    Text(text = "Hi, my name is $name!")
}
```

Le conteneur fourni par défaut est **Box**, mais vous pouvez le remplacer par un autre type de conteneur.

```
@Composable
fun Greeting(name: String) {
    Box { this: BoxScope
        Text(text = "Hello $name!")
    }
}
```

3. Supprimez **Box** et saisissez **Surface()** à la place.

```
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
    Surface() {
        Text(
            text = "Hi, my name is $name!",
            modifier = modifier
        )
    }
}
```

4. Le conteneur **Surface** comporte un paramètre **color**. Définissez-le sur **Color** :

```
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
    Surface(color = Color) {
        Text(
            text = "Hi, my name is $name!",
            modifier = modifier
        )
    }
}
```

5. Lorsque vous saisissez "**Color**" (Couleur), vous remarquerez peut-être que cet élément est en rouge et qu'il est souligné. Pour résoudre ce problème, faites défiler l'écran jusqu'en haut du fichier, où est indiquée l'instruction "**import**", puis appuyez sur les points de suspension.



`import` ...

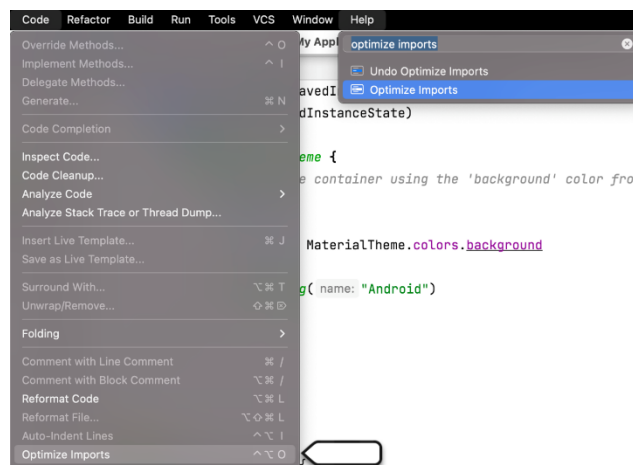
6. Ajoutez cette instruction en bas de la liste des **importations**.

```
import androidx.compose.ui.graphics.Color
```

La liste complète des importations se présente alors comme suit :

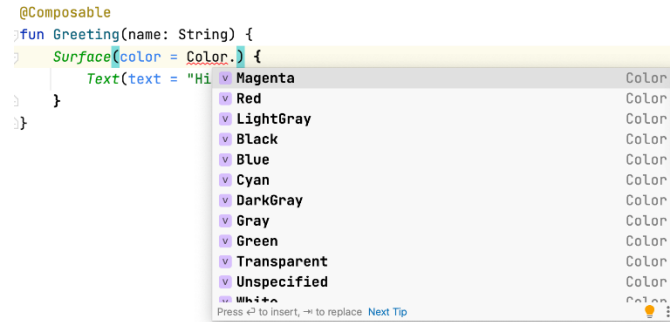
```
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.activity.enableEdgeToEdge
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.padding
import androidx.compose.material3.Scaffold
import androidx.compose.material3.Surface
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.tooling.preview.Preview
import com.example.greetingcard.ui.theme.GreetingCardTheme
```

7. Dans votre code, il est recommandé de classer les importations par ordre alphabétique. Pour ce faire, appuyez sur **Aide** dans la barre d'outils supérieure, saisissez **Optimiser les importations**, puis cliquez sur **Optimiser les importations**.



8. Notez que la couleur que vous avez saisie entre parenthèses pour **Surface** est maintenant **soulignée en rouge** (et non plus rouge et soulignée). Pour résoudre ce problème, ajoutez un point après la couleur. Un pop-up s'affiche alors et vous propose différentes options de couleur.

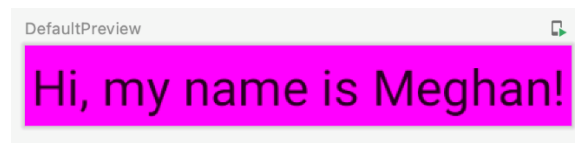
Particulièrement efficace, cette fonctionnalité d'Android Studio vous apportera une aide précieuse dès que l'occasion se présente. Dans ce cas, la fonctionnalité sait que vous souhaitez spécifier une couleur et vous en propose donc plusieurs.



9. Choisissez une couleur pour votre surface. Nous utilisons le **magenta**, mais vous pouvez choisir la couleur que vous préférez.

```
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
    Surface(color = Color.Magenta) {
        Text(
            text = "Hi, my name is $name!",
            modifier = modifier
        )
    }
}
```

10. Cliquez sur **Compiler et actualiser**. Votre texte est entouré par la couleur que vous avez choisie.



7. Ajouter une marge intérieure

Maintenant que votre texte a une couleur d'arrière-plan, vous allez ajouter de l'espace (une marge intérieure) autour du texte.

Un élément [Modifier](#) permet d'améliorer ou de décorer un composable. Vous pouvez utiliser le modificateur **padding**, qui applique un espace autour de l'élément (dans ce cas, il s'agit d'ajouter de l'espace autour du texte). Pour ce faire, utilisez la fonction [Modifier.padding\(\)](#).

1. Ajoutez ces importations dans la section des instructions d'importation.

Assurez-vous d'utiliser l'option **Optimiser les importations** pour classer les nouvelles importations par ordre alphabétique.

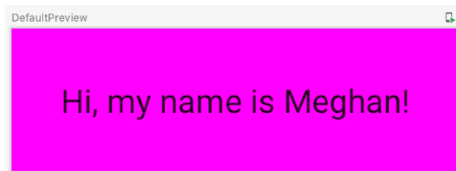
```
import androidx.compose.ui.unit.dp
import androidx.compose.foundation.layout.padding
```

2. Ajoutez un modificateur de marge intérieure au texte avec une taille de **24.dp**, puis cliquez sur **Compiler et actualiser**.

Remarque : Vous en apprendrez davantage sur les pixels indépendants de la densité dans le parcours suivant. Si vous souhaitez en savoir plus maintenant, consultez plutôt [cet article](#).



```
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
    Surface(color = Color.Magenta) {
        Text(
            text = "Hi, my name is $name!",
            modifier = modifier.padding(24.dp)
        )
    }
}
```



Félicitations ! Vous avez créé votre première application Android dans Compose. Personnalisez maintenant votre application en changeant de couleur et en saisissant d'autres textes.

8. Conclusion

Vous avez découvert Android Studio et vous avez créé votre première application Android avec Compose.

Résumé

- Pour créer un projet : ouvrez Android Studio, cliquez sur **Nouveau projet > Activité Compose vide > Suivant**, saisissez le nom de votre projet, puis configurez ses paramètres.
- Pour voir à quoi ressemble votre application, utilisez le volet **Aperçu**.
- Les **fonctions modulables** sont semblables aux fonctions standards, à quelques différences près : la première lettre de leur nom est mise en majuscule, l'annotation `@Composable` est ajoutée avant la fonction et les fonctions `@Composable` ne peuvent rien renvoyer.
- Un élément `Modifier` permet d'améliorer ou de décorer un composable.