Animations in Compose Cheat Sheet

Animate appearing / disappearing 🧞

Use **AnimatedVisibility** to hide/show a Composable.
👧🏾👦🏼 Children inside **AnimatedVisibility** can use **Modifier.animateEnterExit()** for their own enter/exit transition.

```
AnimatedVisibility(visible) {
    // your composable here
}
```

⚠️ *Adds / Removes the item from composition.*

OR

```
val animatedAlpha = animateFloatAsState(/*your
    changing value here*/)
Column(modifier = Modifier.graphicsLayer{
    alpha = animatedAlpha.value
}) {
    // your composable here
}
```

⚠️ *Keeps item in the composition - animates its alpha instead*

Animate size changes 🐣➡️🐥

Use **animateContentSize** for animations between composable size changes.

```
var message by remember { mutableStateOf("Hello") }
Box(
    modifier = Modifier
        .background(Color.Blue)
        .animateContentSize()
) {
    Text(text = message)
}
```

⚠️ *Ordering of the modifier matters, make sure to place it before any size modifiers.*

Animate individual properties from state 🎛️

Use **animate*AsState** APIs for any property animations based on state (for instance: state from a ViewModel). Use the variable in a Modifier or Canvas drawing properties.

```
val animatedColor = animateColorAsState(/*your
    changing value here*/)
Column(modifier = Modifier.drawBehind {
    drawRect(animatedColor.value)
}) {
    // your composable here
}

animateDpAsState()
animateOffsetAsState()
animateFloatAsState()
animateSizeAsState()
animateRectAsState()
animateIntAsState()
```

Animated Vector Drawable 〰️

Animate VectorDrawable paths with **animatedVectorResource**

```
val image = AnimatedImageVector
    .animatedVectorResource(R.drawable.avd_hourglass)

var atEnd by remember { mutableStateOf(false) }
Image(
    painter = rememberAnimatedVectorPainter(image, atEnd),
    modifier = Modifier.clickable {
       atEnd = !atEnd
    },
    contentScale = ContentScale.Crop,
    contentDescription = "hourglass"
)
```

Lazy list item changes 🦥

Animate item reordering of items in a list use **animateItemPlacement().**

```
LazyColumn {
    items(books, key = { it.id }) {
        Row(Modifier.animateItemPlacement(
            tween(durationMillis = 250)
        )) {
            // ...
        }
    }
}
```

⚠️ *Make sure to specify a key for the correct replacement.*
🔜 *Additions and deletions are coming soon.*

Animate changes between Composables ✨

Change between different Composables based on state changes using **AnimatedContent**. For a simple fade between the two, use **CrossFade** instead.

```
AnimatedContent(state) { targetState ->
    when (targetState) {
        Loaded -> /* your composable */
        Loading -> /* your composable */
    }
}
```

Animate multiple properties at once 💞

Use the **Transition** API to animate multiple properties at the same time when transitioning between different states.

```
var currentState by remember { mutableStateOf(Collapsed) }
val transition = updateTransition(currentState)

val rect by transition.animateRect { state ->
    when (state) {
        Collapsed -> Rect(0f, 0f, 100f, 100f)
        Expanded -> Rect(100f, 100f, 300f, 300f)
    }
}
val borderWidth by transition.animateDp { state ->
    when (state) {
        Collapsed -> 1.dp
        Expanded -> 0.dp
    }
}
```

Repeat an animation 🔁

Use **infiniteRepeatable** to continuously repeat your animation. Change **RepeatMode**'s to specify how it should go back and forth.

Use **finiteRepeatable** to repeat a set number of times.

```
val infiniteTransition = rememberInfiniteTransition()
val color by infiniteTransition.animateColor(
    initialValue = Color.Red,
    targetValue = Color.Green,
    animationSpec = infiniteRepeatable(
        animation = tween(1000, easing = LinearEasing),
        repeatMode = RepeatMode.Reverse
    )
)

Box(Modifier.fillMaxSize().background(color))
```

Start an animation on launch 🏁

LaunchedEffect is run when a Composable enters the composition.

```
val alphaAnimation = remember {
    Animatable(0f)
}
LaunchedEffect(key) {
    alphaAnimation.animateTo(1f)
}
Box(modifier = Modifier.alpha(alphaAnimation))
```

⚠️ *If used in a lazy layout, LaunchedEffect will be called every time you scroll your view on and off screen. You may need to **hoist your state** outside of the lazy composable to have the animation only run once.*

Sequential animations

Use the **Animatable** coroutine APIs to do sequential animations.

Calling **animateTo** on the Animatable one after the other will wait for the previous animations to finish before proceeding to the next as its a suspend function.

```
val alphaAnimation = remember { Animatable(0f) }
val yAnimation = remember { Animatable(0f) }

LaunchedEffect("animationKey") {
    alphaAnimation.animateTo(1f)
    yAnimation.animateTo(100f)
    yAnimation.animateTo(500f, animationSpec = tween(100))
}
```

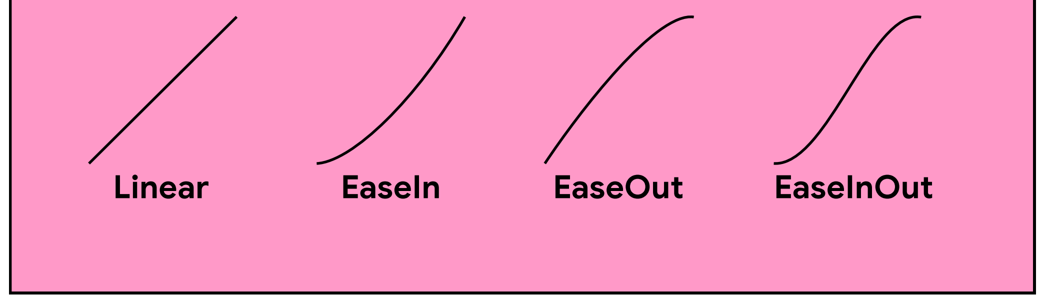
Animation specs 👓

Specify how animation value should transform between the start and target values:

✋ **tween**- animate (with easing) be**tween** two values with a duration.
🎾 **spring** - physics-based animation with damping ratio and stiffness (no duration)
🔑 **keyframes** - spec for specifying different specs at different key frames of the animation.
🔁 **repeatable**- duration based spec runs repeatedly for # of iterations.
🔁 **infiniteRepeatable** - duration based spec runs forever
🫰 **snap** - spec that instantly switches to new value

Easing functions 🎢

Describe the rate of change over time for an animation. **Linear** moves at the same constant speed. Others like **EaseIn**, are slow to start then progress to a linear function.



Concurrent animations

Use coroutine APIs, or Transition API (see transition block for alternative) for concurrent animations.

Using launch in a coroutine context will launch the animations at the same time.

```
val alphaAnimation = remember { Animatable(0f) }
val yAnimation = remember { Animatable(0f) }

LaunchedEffect(key) {
    launch {
        alphaAnimation.animateTo(1f)
    }
    launch {
        yAnimation.animateTo(100f)
    }
}
```

Learn more 📚

docs: goo.gle/compose-animation
codelab: goo.gle/compose-animation-codelab

#JetpackCompose