















#drivindconrtaily

@Elizadarnkoder



**animata\*Assate**

## animateas

-  `animateFloatAsState` (*targetValue: Float, ...*)      `State<Float>`
-  `animateColorAsState` (*targetValue: Color, ...*)      `State<Color>`
-  `animateDpAsState` (*targetValue: Dp, ...*)      `State<Dp>`
-  `animateIntAsState` (*targetValue: Int, ...*)      `State<Int>`
-  `animateOffsetAsState` (*targetValue: Offset, ...*)      `State<Offset>`
-  `animateRectAsState` (*targetValue: Rect, ...*)      `State<Rect>`
-  `animateSizeAsState` (*targetValue: Size, ...*)      `State<Size>`
-  `animateValueAsState` (*targetValue: T, typeConverter...*)      `State<T>`
-  `animateIntOffsetAsState` (*targetValue: IntOffset, ...*)      `State<IntOffset>`
-  `animateIntSizeAsState` (*targetValue: IntSize, ...*)      `State<IntSize>`





## animate\*AsState

### animateas

```
f animateFloatAsState(targetValue: Float, ...)    State<Float>
f animateColorAsState(targetValue: Color, ...)    State<Color>
f animateDpAsState(targetValue: Dp, ...)          State<Dp>
f animateIntAsState(targetValue: Int, ...)         State<Int>
f animateOffsetAsState(targetValue: Offset, ...)   State<Offset>
f animateRectAsState(targetValue: Rect, ...)       State<Rect>
f animateSizeAsState(targetValue: Size, ...)       State<Size>
f animateValueAsState(targetValue: T, typeConverter... State<T>
f animateIntOffsetAsState(targetValue: Int0...     State<IntOffset>
f animateIntSizeAsState(targetValue: IntSize,...   State<IntSize>
```

## animate\*AsState

```
@Composable
fun Heart() {
    var animated by remember { mutableStateOf(false) }
    val color = if (animated) Color.Red else Color(0xFFFFB79A8)

    ...

    Image(
        ...
        colorFilter = ColorFilter.tint(color)
    )
}
```