- Animations in Flutter are a powerful way to create dynamic and engaging user interfaces. Here are the basics of animations in Flutter:

AnimationController:

- The AnimationController is the heart of the animation. It's responsible for controlling the duration, playback status (forward, reverse), and more.

```
AnimationController controller = AnimationController(
  duration: Duration(seconds: 2),
  vsync: this, // TickerProvider, often provided by StatefulWidget
);
```

Tween:

- A Tween defines the range of values that the animation will interpolate. It specifies the starting and ending values.

```
Tween<double> opacityTween = Tween<double>(begin: 0.0, end: 1.0);
```

Curves:

- Curves determine the timing of the animation. They define how the values change over time. Flutter provides various built-in curves, and you can create custom ones.

```
CurvedAnimation curvedAnimation = CurvedAnimation(
  parent: controller,
  curve: Curves.easeInOut,
);
```

Animation:

- The Animation class represents the evolving state of an animation. It's often created by applying a Tween to an AnimationController.

```
Animation<double> opacityAnimation = opacityTween.animate(controller);
```

Listeners and Builders:

- Listeners are callbacks that get invoked when the animation value changes. They are handy for updating the UI based on the animation's progress.

```
controller.addListener(() {
  // Update UI based on the current animation value
  print(opacityAnimation.value);
});
```

The AnimatedBuilder widget simplifies UI updates by automatically rebuilding when the animation changes.

```
AnimatedBuilder(
  animation: opacityAnimation,
  builder: (context, child) {
    return Opacity(
      opacity: opacityAnimation.value,
      child: // Your animated widget here,
    );
  },
);
```

Dispose:

- Always dispose of animation controllers when they are no longer needed to prevent memory leaks.

```
@override
void dispose() {
  controller.dispose();
  super.dispose();
}
```

These basics provide a foundation for creating various animations in Flutter. Whether you're working with opacity, size, position, or other properties, these principles remain consistent across different types of animations.

what types of animations in dart with examples

Implicit Animations:

- Implicit animations automatically animate changes to a property over time. Flutter provides several built-in implicit animations, such as AnimatedContainer, AnimatedOpacity, and AnimatedPositioned.
- Here's an example using AnimatedContainer:

```dart
import 'package:flutter/material.dart';

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: MyAnimatedContainer(),
    );
  }
}

class MyAnimatedContainer extends StatefulWidget {
  @override
  _MyAnimatedContainerState createState() => _MyAnimatedContainerState();
}

class _MyAnimatedContainerState extends State<MyAnimatedContainer> {
  bool _isExpanded = false;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Animated Container Example'),
      ),
      body: Center(
        child: GestureDetector(
          onTap: () {
            setState(() {
              _isExpanded = !_isExpanded;
            });
          },
          child: AnimatedContainer(
            duration: Duration(seconds: 1),
            width: _isExpanded ? 200.0 : 100.0,
            height: _isExpanded ? 200.0 : 100.0,
            color: _isExpanded ? Colors.blue : Colors.red,
            child: Center(
              child: Text('Tap to Expand'),
            ),
          ),
        ),
      ),
    );
```

```
    }
}
```

Tween Animations:

- Tween animations interpolate between two values over time. The Tween class is commonly used for this purpose. Here's an example of a simple Tween animation:

```dart
import 'package:flutter/material.dart';

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: MyTweenAnimation(),
    );
  }
}

class MyTweenAnimation extends StatefulWidget {
  @override
  _MyTweenAnimationState createState() => _MyTweenAnimationState();
}

class _MyTweenAnimationState extends State<MyTweenAnimation> with SingleTickerProviderStateMixin {
  late AnimationController _controller;
  late Animation<double> _animation;

  @override
  void initState() {
    super.initState();

    _controller = AnimationController(
      duration: Duration(seconds: 2),
      vsync: this,
    );

    _animation = Tween<double>(begin: 0, end: 300).animate(_controller);

    _controller.forward();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Tween Animation Example'),
      ),
      body: Center(
        child: AnimatedBuilder(
          animation: _animation,
          builder: (context, child) {
            return Container(
```

```
              width: _animation.value,
              height: _animation.value,
              color: Colors.blue,
              child: Center(
                child: Text('Tween Animation'),
              ),
            );
          },
        ),
      ),
    );
  }

  @override
  void dispose() {
    _controller.dispose();
    super.dispose();
  }
}
```

Custom Animations:

- For more complex animations, you can create custom animations using the Animation class. This involves defining your own interpolation logic and updating the UI accordingly. Custom animations are often implemented using the AnimationController and Tween classes.

```dart
import 'package:flutter/material.dart';

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: MyCustomAnimation(),
    );
  }
}

class MyCustomAnimation extends StatefulWidget {
  @override
  _MyCustomAnimationState createState() => _MyCustomAnimationState();
}

class _MyCustomAnimationState extends State<MyCustomAnimation> with SingleTickerProviderStateMixin {
  late AnimationController _controller;
  late Animation<double> _animation;

  @override
  void initState() {
    super.initState();

    _controller = AnimationController(
      duration: Duration(seconds: 2),
      vsync: this,
    );

    _animation = Tween<double>(begin: 0, end: 300).animate(_controller);

    _controller.forward();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Custom Animation Example'),
      ),
      body: Center(
        child: AnimatedBuilder(
          animation: _animation,
          builder: (context, child) {
            return Transform.translate(
```

```
              offset: Offset(_animation.value, 0),
              child: Container(
                width: 100.0,
                height: 100.0,
                color: Colors.green,
                child: Center(
                  child: Text('Custom Animation'),
                ),
              ),
            );
          },
        ),
      ),
    );
  }

  @override
  void dispose() {
    _controller.dispose();
    super.dispose();
  }
}
```

- These are just a few examples of the types of animations you can implement in Dart and Flutter. Depending on your needs, you may choose different animation techniques for different scenarios.