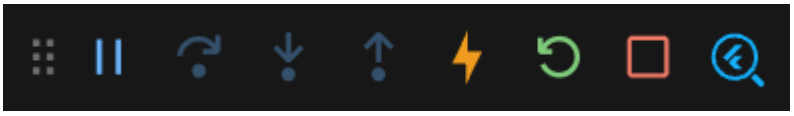
5.2.1 Logging

Aufgaben

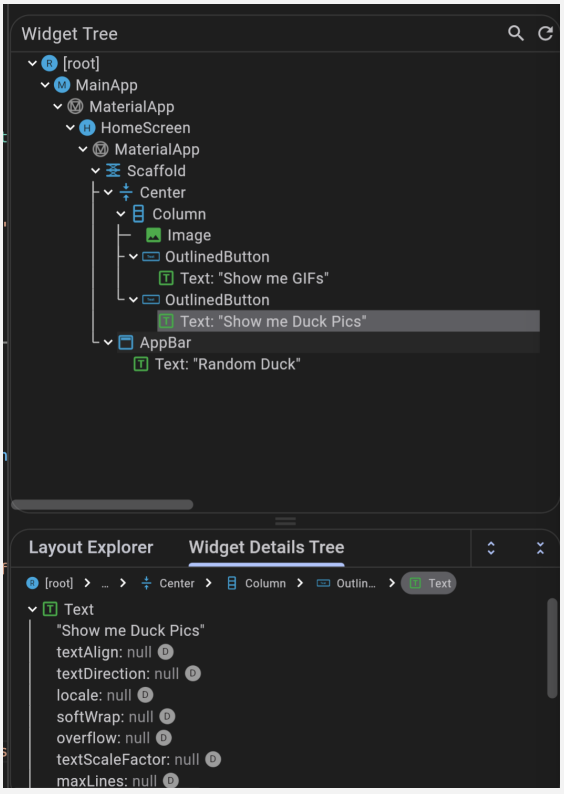
Aufgabe 1

Öffne eine deiner bisherigen Apps und gehe in die Flutter-Dev-Tools (ganz rechts).



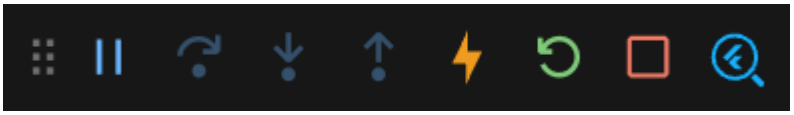
Welche Werte kannst du einem Widget deiner Wahl entnehmen? Notiere.

Deine Antwort:



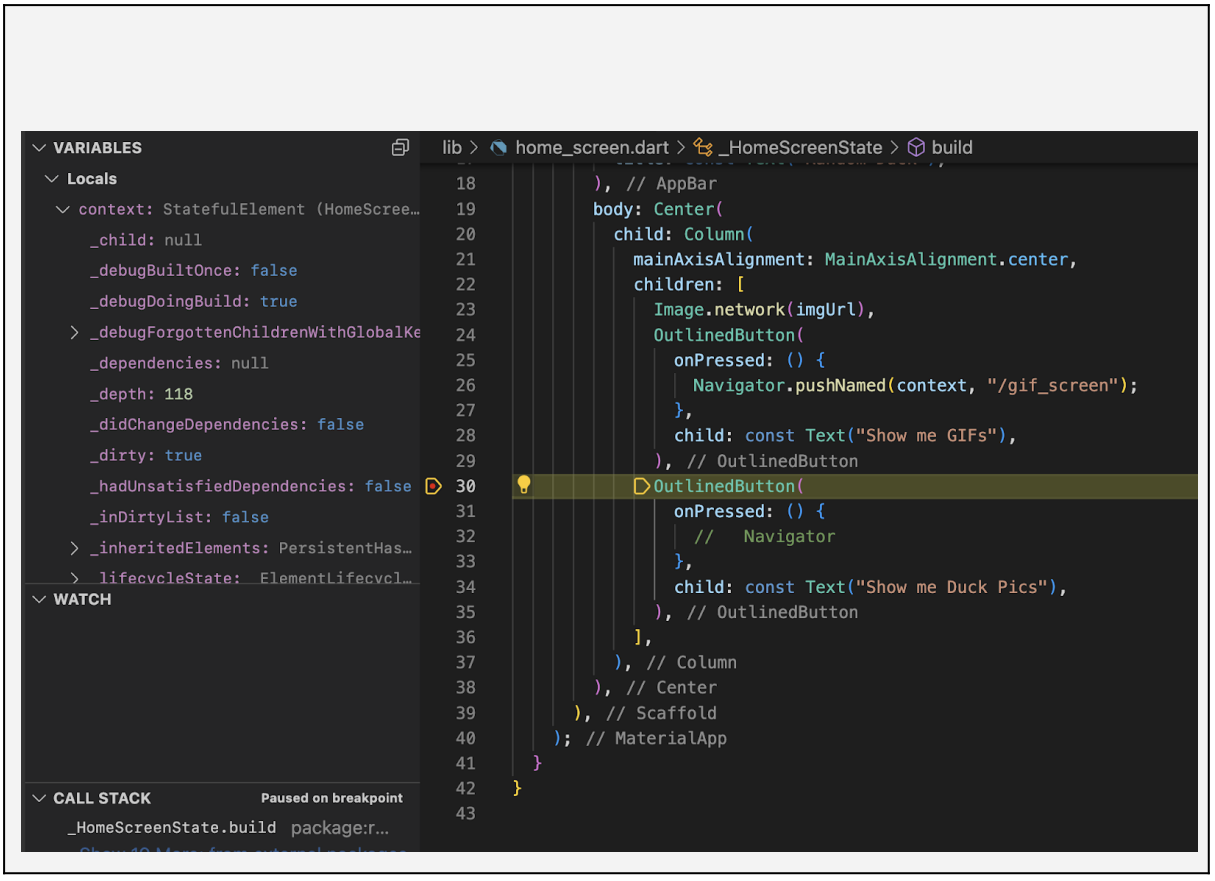
Aufgabe 2

Setze nun einen Breakpoint, sodass die App bei Ausführung dort hängen bleibt.



Poste einen Screenshot, wie die App hängen geblieben ist, und gib einen Wert an, der links im Debug-Fenster zu sehen ist.

Deine Antwort:



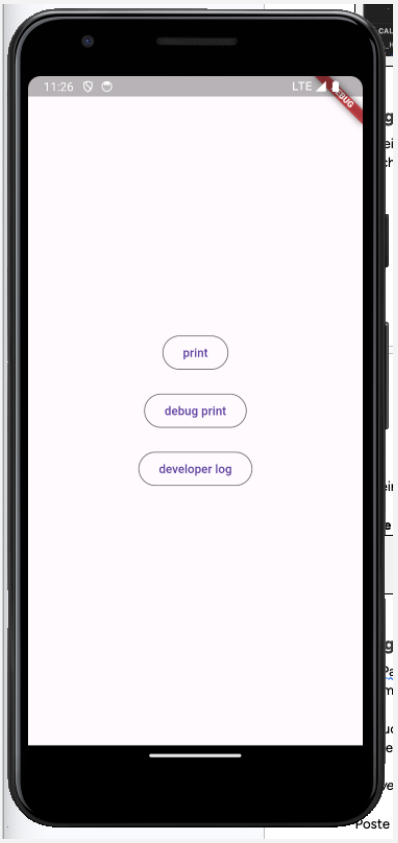
Aufgabe 3

Schreibe eine kleine App. Diese soll verschiedene Buttons haben und jeweils mit verschiedenen Methoden auf die Konsole printen:

1. **print**
2. **debugPrint**
3. **log** (aus dem **developer** package, in Flutter integriert)

Gib einen Screenshot der Ausgabe(n) an.

Deine Antwort:



Aufgabe 4

Das Package **logger** hat verschiedene Log-Level. Schreibe eine kleine App, um damit herumzuspielen und verschiedenfarbige Nachrichten auf der Debugkonsole zu erzeugen.

Versuche auch mal, eine Liste oder sogar Map zu übergeben und schaue, wie das Ergebnis aussieht.

Hinweis: Hierbei kann dir die Doku helfen: https://pub.dev/packages/logger

Poste einen Screenshot der Ergebnisse.

Deine Antwort:

```
import 'dart:developer';

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

```dart
import 'package:logger/logger.dart';

void main() {
 runApp(const MainApp());
}

class MainApp extends StatefulWidget {
 const MainApp({super.key});

 @override
 State<MainApp> createState() => _MainAppState();
}

class _MainAppState extends State<MainApp> {
 // Logger
 var logger = Logger(
   filter: null,
   printer: PrettyPrinter(),
   output: null,
 );

 List<String> logList = ["this", "is", "the", "amazing", "loglist"];

 @override
 Widget build(BuildContext context) {
   return MaterialApp(
     home: Scaffold(
       body: Center(
         child: Column(
           mainAxisAlignment: MainAxisAlignment.center,
           children: [
             // print
             OutlinedButton(
                 onPressed: () {
                   print("Print-Button fired!");
                 },
                 child: const Text("print")),
             const SizedBox(height: 20),

             // debug print
             OutlinedButton(
                 onPressed: () {
                   debugPrint("DebugPrint-Button fired!");
                 },
```

```
                child: const Text("debug print")),
            const SizedBox(height: 20),

            // developer-log
            OutlinedButton(
                onPressed: () {
                  log("DeveloperLog-Button fired!");
                },
                child: const Text("developer log")),
            const SizedBox(height: 20),

            // log
            OutlinedButton(
                onPressed: () {
                  logger.i("logging info text...");
                },
                child: const Text("pretty-log")),

            const SizedBox(height: 20),

            OutlinedButton(
                onPressed: () {
                  logger.e("error log", error: "Test Error");
                },
                child: const Text("Log an error")),

            const SizedBox(height: 20),

            OutlinedButton(
                onPressed: () {
                  logWarningWithList(logList);
                },
                child: const Text("Log a warning"))
          ],
        ),
      ),
    ),
  );
}

void logWarningWithList(List<String> logList) {
  logger.w(logList);
}
}
```

Aufgabe 5

Beschreibe in eigenen Worten, wofür Linting verwendet wird, und wann es mehr oder weniger Sinn macht. Denke auch an die spätere Arbeitswelt.

Deine Antwort:

Linting (“entfusseln) wird verwendet um clean code ansätze bzw standards umzusetzen. Ein Beispiel ist das man const vor einem Konstanten Wert schreibt um die Performanz der App zu verbessern.
In der späteren Arbeitswelt macht es definitv Sinn gemeinsame Standards zu haben sind die Standards mit dem Linting vereinbar macht es definitv Sinn.

Viel Erfolg bei der Bearbeitung der Aufgaben! 😻