

```

@override
Widget build(BuildContext context) {
  // SharedPreferences에서 온보딩 완료 여부 조회
  // isOnboarded에 해당하는 값이 null을 반환하는 경우 false 할당
  bool isOnboarded = prefs.getBool("isOnboarded") ?? false;
  return MaterialApp(
    debugShowCheckedModeBanner: false,
    theme: ThemeData(
      textTheme: GoogleFonts.getTextTheme('Jua'),
    ),
    home: isOnboarded ? HomePage() : OnboardingPage(),
  );
}

class OnboardingPage extends StatelessWidget {
  const OnboardingPage({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: IntroductionScreen(
        pages: [
          // 첫 번째 페이지
          PageViewModel(
            title: "빠른 개발",
            body: "Flutter의 hot reload는 쉽고 UI 빌드를 도와줍니다.",
            image: Padding(
              padding: EdgeInsets.all(32),
              child: Image.network(
                'https://user-images.githubusercontent.com/26322627/143761841-ba5c8fa6-af01-4740-81b8-b8ff23d40253.png'),
            ),
            decoration: PageDecoration(
              titleTextStyle: TextStyle(
                color: Colors.blueAccent,
                fontSize: 24,
                fontWeight: FontWeight.bold,
              ),
              bodyTextStyle: TextStyle(
                color: Colors.black,
                fontSize: 18,
              ),
            ),
          ),
          // 두 번째 페이지
          PageViewModel(
            title: "표현력 있고 유연한 UI",
            body: "Flutter에 내장된 아름다운 위젯들로 사용자를 기쁘게 하세요.",
            image: Image.network(
              'https://user-images.githubusercontent.com/26322627/143762620-8cc627ce-62b5-426b-bc81-a8f578e8549c.png'),
            decoration: PageDecoration(
              titleTextStyle: TextStyle(
                color: Colors.blueAccent,
                fontSize: 24,
                fontWeight: FontWeight.bold,
              ),
              bodyTextStyle: TextStyle(
                color: Colors.black,
                fontSize: 18,
              ),
            ),
          ),
        ],
        next: Text("Next", style: TextStyle(fontWeight: FontWeight.w600)),
        done: Text("Done", style: TextStyle(fontWeight: FontWeight.w600)),
        onDone: () {
          // Done 클릭시 isOnboarded = true로 저장
          prefs.setBool("isOnboarded", true);

          // Done 클릭시 페이지 이동
          Navigator.pushReplacement(
            context,
            MaterialPageRoute(builder: (context) => HomePage()),
          );
        },
      ),
    );
  }
}

```

```

}

class HomePage extends StatelessWidget {
  const HomePage({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("HomePage"),
        actions: [
          // 삭제 버튼
          IconButton(
            onPressed: () {
              // SharedPreferences에 저장된 모든 데이터 삭제
              prefs.clear();
            },
            icon: Icon(Icons.delete),
          ),
        ],
      ),
      body: Center(
        child: Text(
          "환영합니다.",
          style: TextStyle(fontSize: 28),
        ),
      ),
    );
  }
}

```

### 03. 버킷 리스트 앱 만들기



나만의 버킷리스트를 작성하는 앱을 만들어 보도록 하겠습니다.

#### ▼ 완성본

코드스니펫을 복사한 뒤 주소창에 붙여 넣으면, DartPad로 접속합니다.

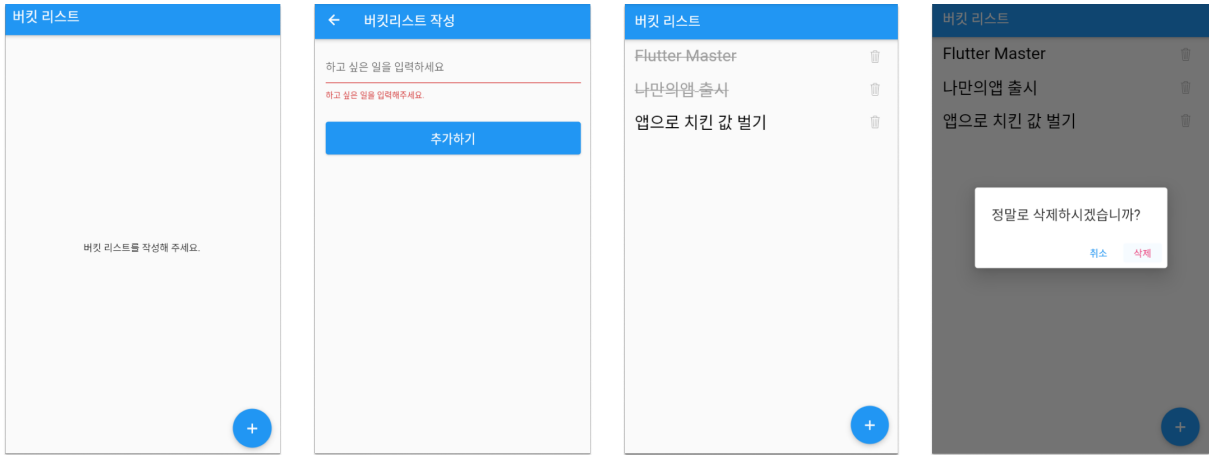
#### ▼ [코드스니펫] DartPad 버킷 리스트 완성본

[https://dartpad.dev/?id=4a6cb05fefdcfe260451e3a7f666acc&null\\_safety=true](https://dartpad.dev/?id=4a6cb05fefdcfe260451e3a7f666acc&null_safety=true)



버킷리스트 앱의 기능은 다음과 같습니다.

1. 버킷 리스트 작성(Create)
2. 버킷 리스트 조회(Read)
3. 버킷 리스트 수정(Update)
4. 버킷 리스트 삭제>Delete)



Create / Read / Update / Delete의 앞 글자를 따서 **CRUD**라고 부릅니다.  
**CRUD**는 가장 기본이 되는 데이터 처리 기능입니다.

게시판 기능을 만든다면 아래 **CRUD** 기능이 필수적으로 제공되어야 합니다.

1. 글 쓰기(Create)
2. 글 읽기(Read)
3. 글 수정(Update)
4. 글 삭제>Delete)

유저 정보를 다루는 과정도 **CRUD**로 표현하면 다음과 같습니다.

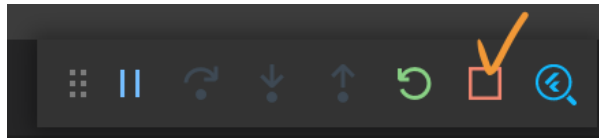
1. 회원 가입(Create)
2. 프로필 보여주기(Read)
3. 회원 정보 수정(Update)
4. 회원 탈퇴>Delete)

이와 같이 **CRUD**는 다루는 데이터의 종류만 바뀌고 항상 기본적으로 구현하는 **데이터 처리 기능**입니다.

## ▼ 1) 프로젝트 준비

### ▼ Flutter 프로젝트 생성

1. VSCode에서 아래와 같이 네모 모양의 **Stop** 버튼을 눌러 기존에 실행한 앱을 종료해주세요.



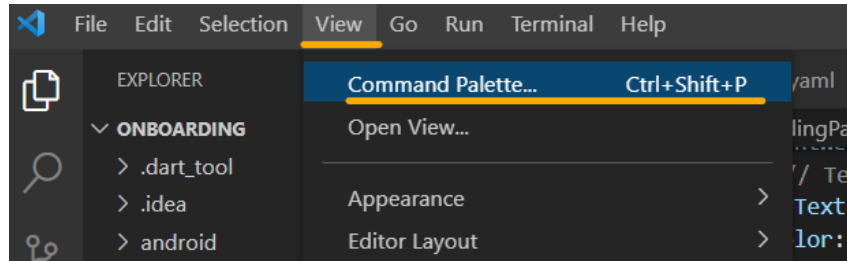
2. **View** → **Command Palette**를 선택해주세요.



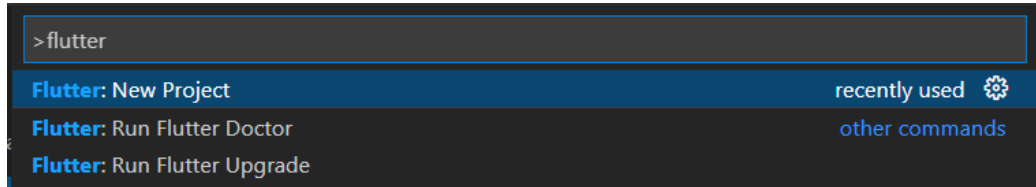
**Command Palette** 단축키

window : **Ctrl + Shift + P**

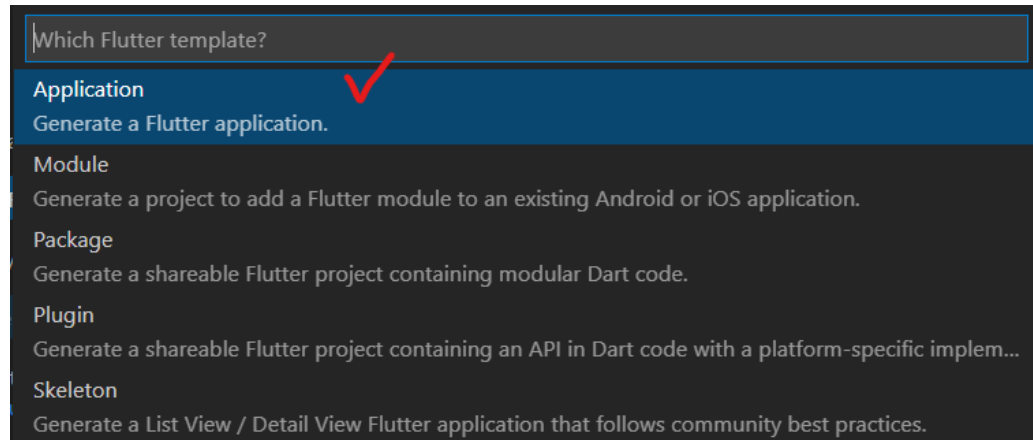
macOS : **Cmd + Shift + P**



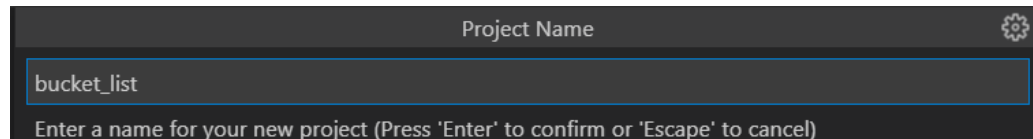
- 명령어를 검색하는 팝업창이 뜨면, `flutter` 라고 입력한 뒤 `Flutter: New Project` 를 선택해주세요.



- `Application` 을 선택해주세요.



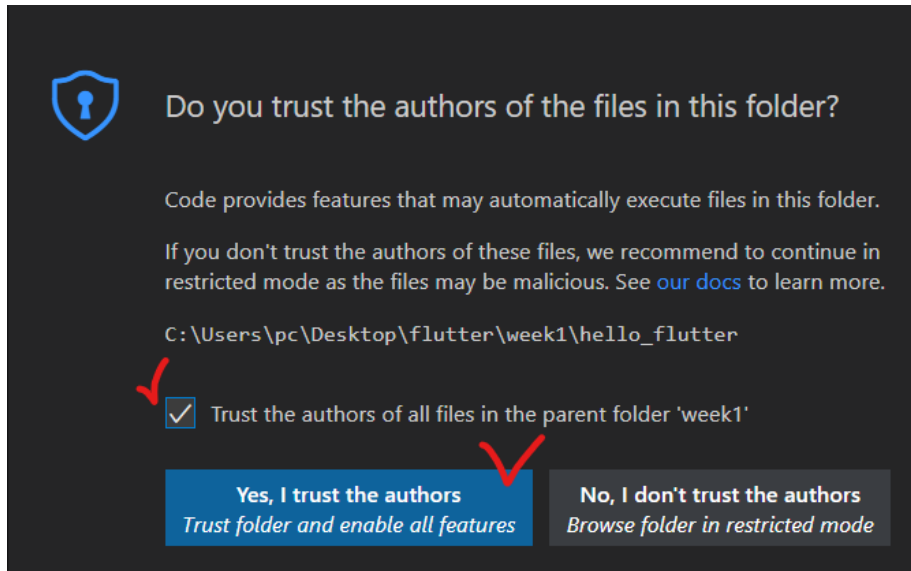
- 프로젝트를 저장할 폴더를 선택하는 화면이 나오면 `flutter` 폴더를 선택한 뒤 `Select a folder to create the project in` 버튼을 눌러 주세요.
- 프로젝트 이름을 `bucket_list` 로 입력해주세요.



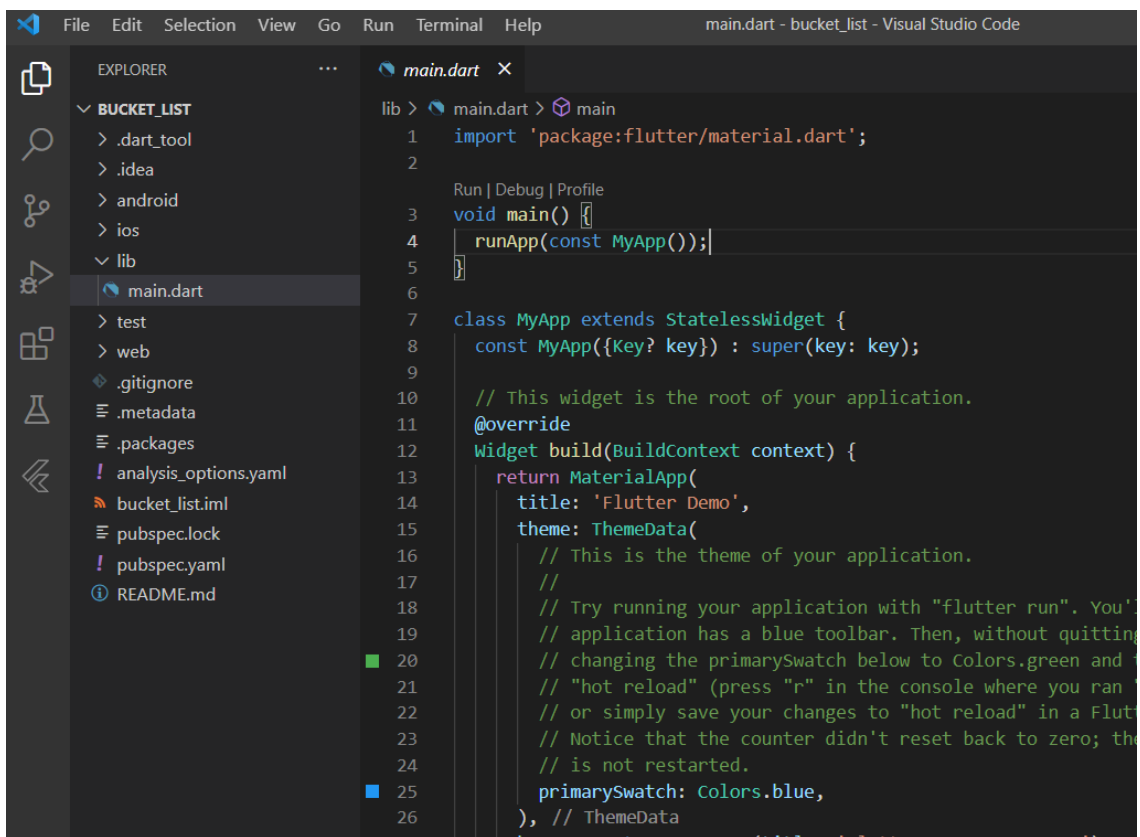
만약 중간에 아래와 같은 팝업이 뜬다면, 체크박스를 선택한 뒤 파란 버튼을 클릭해주세요. (팝업이 안보이시면 넘어가주세요!)



아래 팝업에 대한 자세한 사항은 [링크](#)를 참고해주세요.



7. 다음과 같이 프로젝트가 생성됩니다.



8. 불필요한 힌트 숨기기

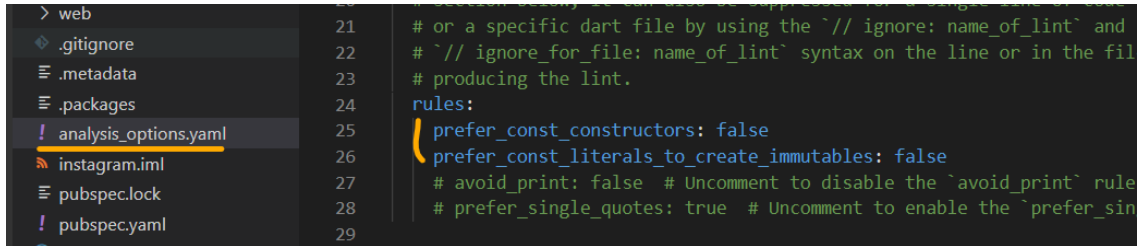
코드스니펫을 복사해서 `analysis_options.yaml` 파일의 24번째 라인 뒤에 붙여 넣고 저장해주세요.



아래 내용은 학습 단계에서 불필요한 내용을 화면에 표시하지 않도록 설정하는 과정입니다.

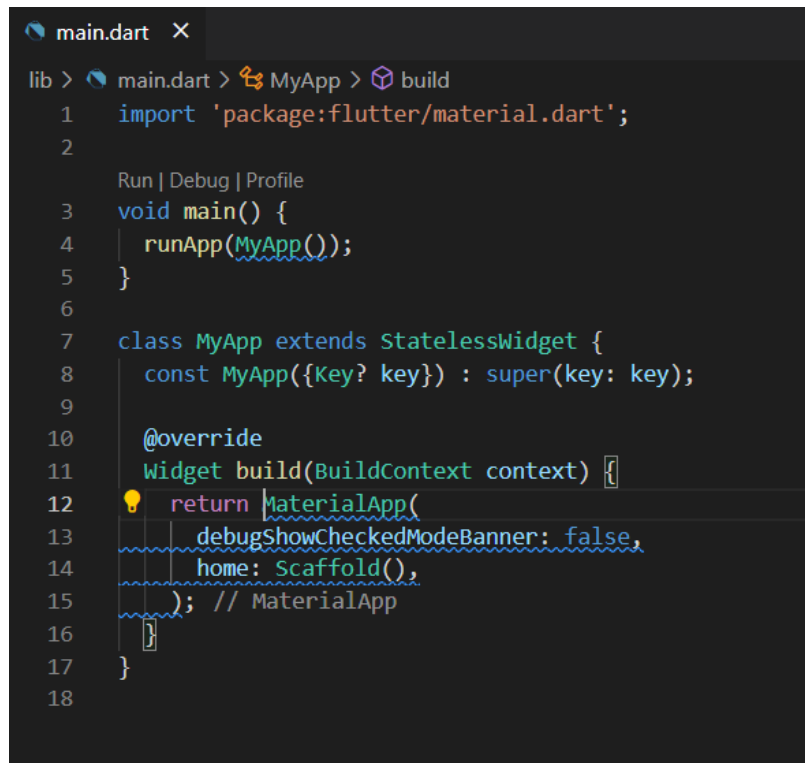
#### ▼ [코드스니펫] analysis\_options.yaml

```
prefer_const_constructors: false
prefer_const_literals_to_create_immutables: false
```



#### ▼ 어떤 의미인지 궁금하신 분들을 위해

- `main.dart` 파일을 열어보시면 파란 실선이 있습니다.



- 파란 줄은, 개선할 여지가 있는 부분을 VSCode가 알려주는 표시입니다.  
12번째 라인에 마우스를 올리면 아래와 같이 설명이 뜹니다.

위젯이 변경될 일이 없기 때문에 `const` 라는 키워드를 앞에 붙여 상수로 선언하라는 힌트입니다.



상수로 만들면 어떤 이점이 있나요?

상수로 선언된 위젯들은 화면을 새로 고침 할 때 해당 위젯들은 변경을 하지 않기 때문에 스킵하여 성능상 이점이 있습니다.

아래와 같이 `Icon` 앞에 `const` 키워드를 붙여주시면 됩니다.

- 지금은 학습 단계이니 눈에 띄지 않도록 해주도록 하겠습니다.

9. 아래 코드스니펫을 복사해서, `main.dart` 의 기존 코드를 모두 지우고 붙여 넣은 뒤 저장해 주세요.

#### ▼ [코드스니펫] main.dart

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

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

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      home: HomePage(),
    );
  }
}
```

```

/// 홈 페이지
class HomePage extends StatelessWidget {
  const HomePage({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("버킷 리스트"),
      ),
      body: Center(child: Text("버킷 리스트를 작성해 주세요.")),
      floatingActionButton: FloatingActionButton(
        child: Icon(Icons.add),
        onPressed: () {
          // + 버튼 클릭시 버킷 생성 페이지로 이동
          Navigator.push(
            context,
            MaterialPageRoute(builder: (_) => CreatePage()),
          );
        },
      ),
    );
  }
}

/// 버킷 생성 페이지
class CreatePage extends StatelessWidget {
  const CreatePage({Key? key}) : super(key: key);

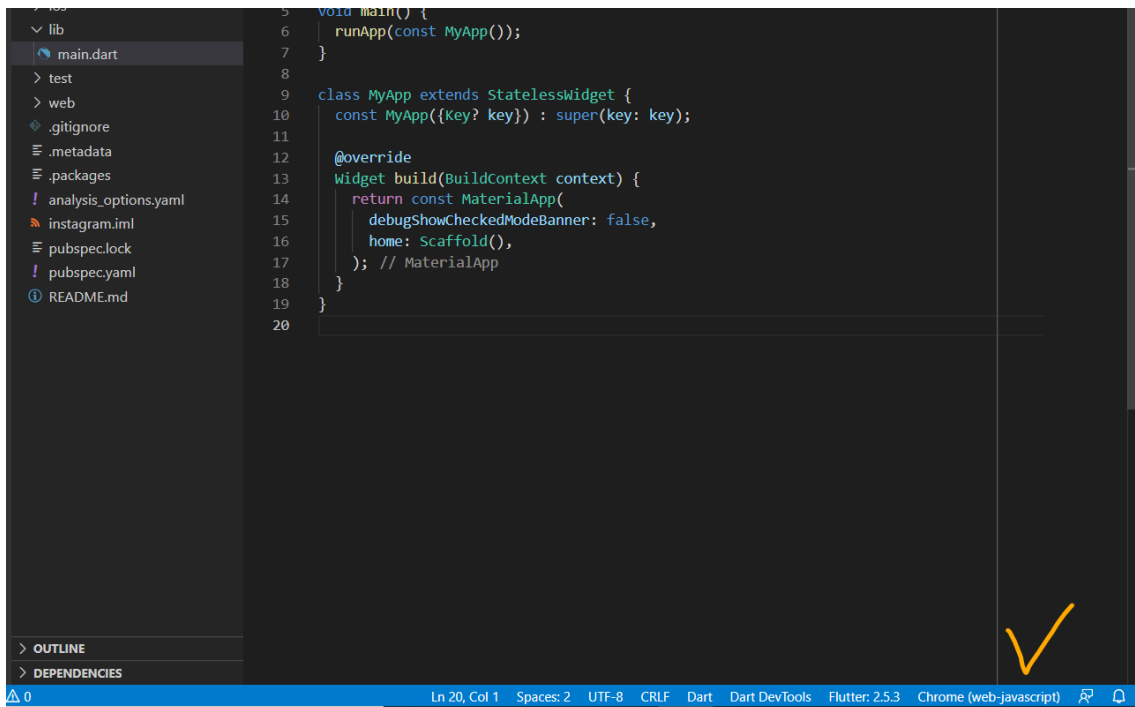
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("버킷리스트 작성"),
        // 뒤로가기 버튼
        leading: IconButton(
          icon: Icon(CupertinoIcons.chevron_back),
          onPressed: () {
            Navigator.pop(context);
          },
        ),
      ),
      body: Padding(
        padding: const EdgeInsets.all(16),
        child: Column(
          children: [
            // 텍스트 입력창
            TextField(
              autofocus: true,
              decoration: InputDecoration(
                hintText: "하고 싶은 일을 입력하세요",
              ),
            ),
            SizedBox(height: 32),
            // 추가하기 버튼
            SizedBox(
              width: double.infinity,
              height: 48,
              child: ElevatedButton(
                child: Text(
                  "추가하기",
                  style: TextStyle(
                    fontSize: 18,
                  ),
                ),
                onPressed: () {
                  // 추가하기 버튼 클릭시
                },
              ),
            ),
          ],
        ),
      ),
    );
  }
}

```

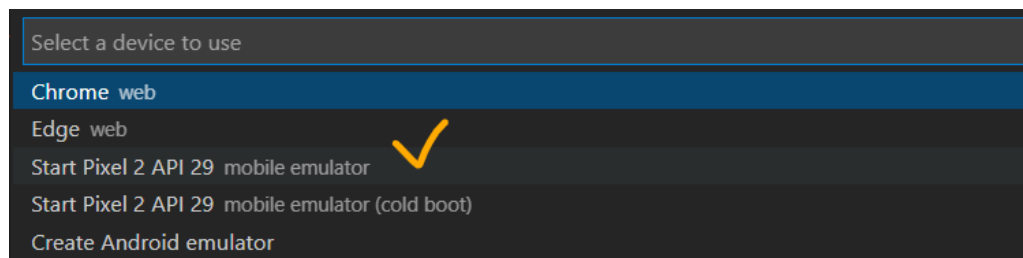


### ▼ 에뮬레이터 실행하기

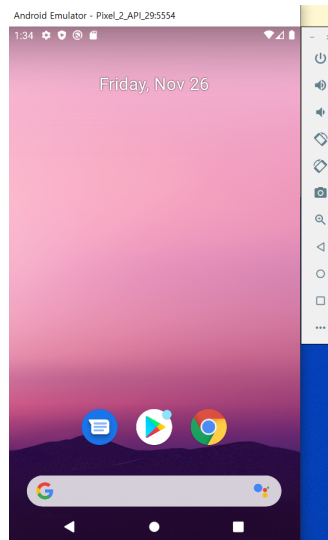
1. VSCode 우측 하단에 **Chrome (web-javascript)** 를 클릭해주세요.  
(에뮬레이터가 이미 실행중이라면 3번으로 이동해 주세요.)



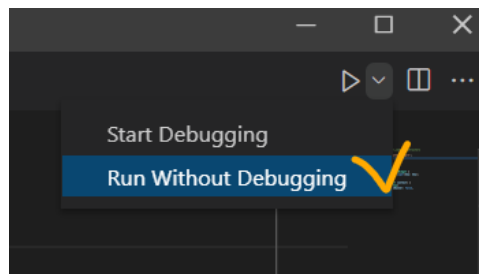
2. **Start Pixel 2 API 29 mobile emulator** 를 선택해주세요. macOS의 경우 iOS 에뮬레이터로 진행하셔도 무방합니다.



잠시 기다리면 에뮬레이터가 실행됩니다.

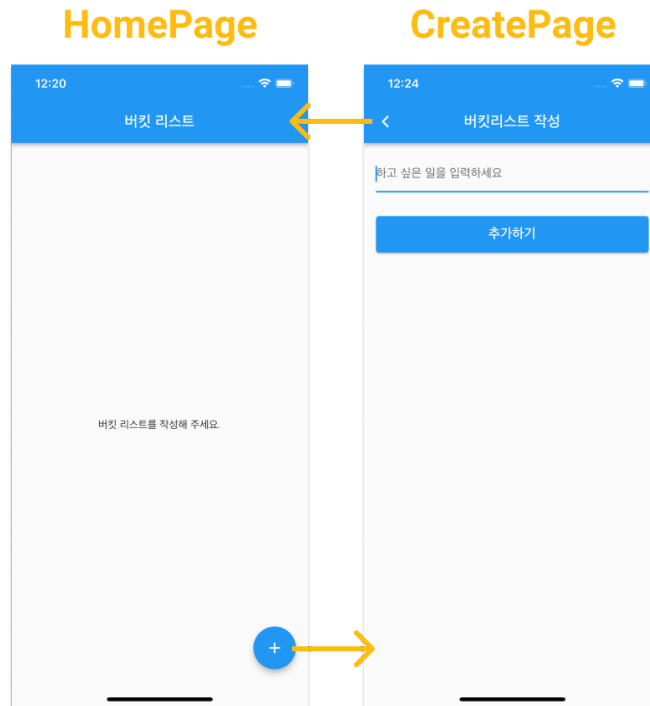


3. VSCode 우측 상단에 **아래 화살표**를 눌러 `Run Without Debugging` 을 눌러주세요.



💡 `Start Debugging` 으로 실행해도 무방합니다. 디버깅 모드는 특정 라인에서 앱 실행을 멈추고 해당 변수에 어떤 값이 들어있는지 볼 수 있지만 `Run without debugging` 이 실행 속도가 더 빨라 안내를 위와 같이 드렸습니다 😊

에뮬레이터에 아래와 같이 버킷 리스트 화면이 나오면 완료!



위와 같이 버킷 리스트를 보여주는 `HomePage` 와 버킷 리스트를 작성하는 `CreatePage` 두 페이지로 구성된 앱입니다.  
기본적인 레이아웃과 페이지 이동 기능은 구현되어 있고, CRUD 기능을 함께 구현해 보도록 하겠습니다.

## ▼ 2) 버킷 리스트 조회(Read)

### 💡 요구사항

- 버킷리스트가 없는 경우, `버킷리스트를 작성해 주세요.` 문구를 보여주기
- 버킷리스트가 있는 경우, 버킷리스트 목록을 보여주기

1. `HomePage` 는 버킷리스트의 유무라는 상태에 따라 다른 화면을 갱신해야 하므로 `StatefulWidget` 으로 변경해줍니다.  
21번째 라인에 `StatelessWidget` 을 클릭한 뒤 Quick Fix(`Ctrl/Cmd + .`)를 누른 뒤 `Convert to StatefulWidget` 을 선택해주세요.

```

18 }
19
20 // 홈 페이지
21 class HomePage extends StatelessWidget {
22   const HomePage({Key? key}) :
23     @override
24

```

Convert to StatefulWidget

아래와 같이 `_HomePageState` 라는 상태 클래스가 추가되었습니다.

```

19
20 /// 홈 페이지
21 class HomePage extends StatefulWidget {
22   const HomePage({Key? key}) : super(key: key);
23
24   @override
25   State<HomePage> createState() => _HomePageState();
26 }
27
28 class _HomePageState extends State<HomePage> {
29   @override
30   Widget build(BuildContext context) {
31     return Scaffold(
32       appBar: AppBar(

```

2. 버킷리스트 가지고 있을 상태 변수 `bucketList` 를 추가해 줍니다.

아래 코드스니펫을 복사해서 28번째 라인 맨 뒤에 붙여 넣어 주세요.

▼ [코드스니펫] HomePage / bucketList 상태 변수

```
List<String> bucketList = ['여행가기']; // 전체 버킷리스트 목록
```



`bucketList` 는 String만 받는 배열로 선언하였습니다.

미리 `bucketList` 에 '여행가기' 라는 항목을 하나 넣어 두었습니다.

```

26 }
27
28 class _HomePageState extends State<HomePage> {
29   List<String> bucketList = ['여행가기']; // 전체 버킷리스트 목록
30
31   @override
32   Widget build(BuildContext context) {
33     return Scaffold(

```

3. `bucketList` 에 항목이 하나라도 있다면, body에 해당 항목이 보여지도록 작성해봅시다.

코드스니펫을 복사해서 37번째 라인을 교체해 주세요.

▼ [코드스니펫] HomePage / bucketList 조건문

```
body: bucketList.isEmpty
  ? Center(child: Text("버킷 리스트를 작성해 주세요."))
  : Center(child: Text('버킷 리스트가 존재합니다!')),
```



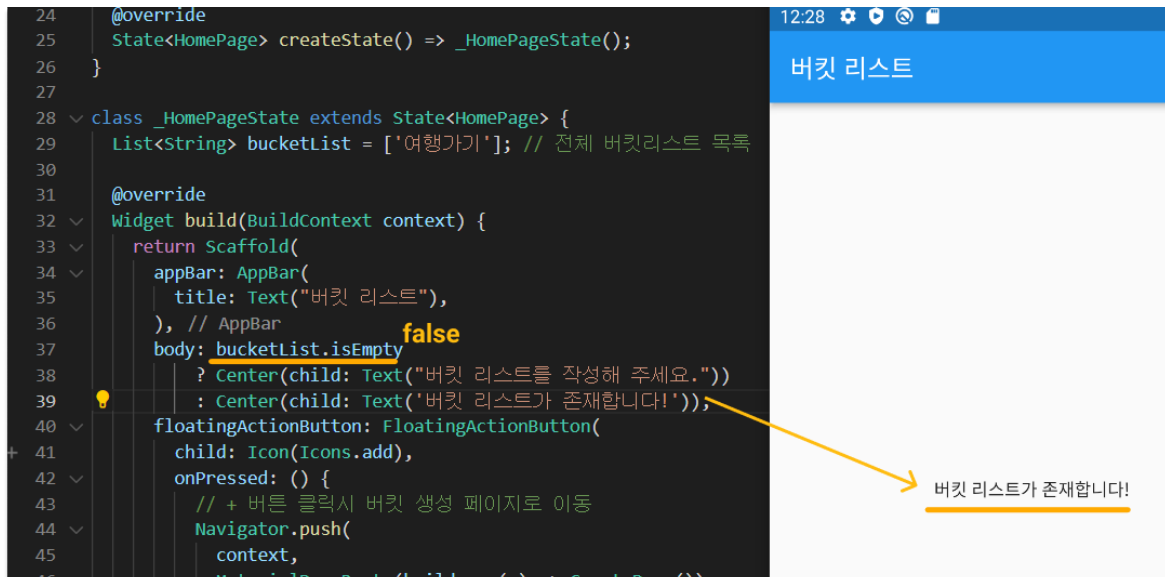
`조건 ? true : false` 형태의 조건문을 사용하여 조건에 따라 위젯을 다르게 보여줄 수 있습니다.

```

34     appBar: AppBar(
35       title: Text("버킷 리스트"),
36     ), // AppBar
37     body: bucketList.isEmpty
38       ? Center(child: Text("버킷 리스트를 작성해 주세요."))
39       : Center(child: Text('버킷 리스트가 존재합니다!')),
40     floatingActionButton: FloatingActionButton(
41       child: Icon(Icons.add),
42       onPressed: () {
43         // + 버튼 클릭시 버킷 생성 페이지로 이동

```

미리 `bucketList` 에 '여행가기' 라는 항목을 하나 넣어 두었기 때문에 `bucketList.isEmpty` 는 `false` 가 되고, '버킷 리스트가 존재합니다!' 문구가 에뮬레이터에 나오는 것을 보실 수 있습니다.



- 버킷리스트가 존재하는 경우, `ListView` 를 이용해 보여주도록 코드스니펫을 복사해서 39번째 라인을 교체해 주세요.

#### ▼ [코드스니펫] HomePage / ListView

```

: ListView.builder(
  itemCount: bucketList.length, // bucketList 개수 만큼 보여주기
  itemBuilder: (context, index) {
    String bucket = bucketList[index]; // index에 해당하는 bucket 가져오기
    return ListTile(
      // 버킷 리스트 할 일
      title: Text(
        bucket,
        style: TextStyle(
          fontSize: 24,
        ),
      ),
      // 삭제 아이콘 버튼
      trailing: IconButton(
        icon: Icon(CupertinoIcons.delete),
        onPressed: () {
          // 삭제 버튼 클릭시
          print('$bucket : 삭제하기');
        },
      ),
      onTap: () {
        // 아이템 클릭시
        print('$bucket : 클릭 됨');
      },
    ),
  ),
)

```

```

    );
  },
),

```

```

37 body: bucketList.isEmpty
38   ? Center(child: Text("버킷 리스트를 작성해 주세요."))
39   : ListView.builder(
40     itemCount: bucketList.length, // bucketList 개수 만큼 보여주기
41     itemBuilder: (context, index) {
42       String bucket = bucketList[index]; // index에 해당하는 bucket 가져오기
43       return ListTile(
44         // 버킷 리스트 할 일
45         title: Text(
46           bucket,
47           style: TextStyle(
48             fontSize: 24,
49           ), // TextStyle
50         ), // Text
51         // 삭제 아이콘 버튼
52         trailing: IconButton(
53           icon: Icon(CupertinoIcons.delete),
54           onPressed: () {
55             // 삭제 버튼 클릭시
56             print('$bucket : 삭제하기');
57           },
58         ), // IconButton
59         onTap: () {
60           // 아이템 클릭시
61           print('$bucket : 클릭 됨');
62         },
63       ); // ListTile
64     },
65   ), // ListView.builder
66   floatingActionButton: FloatingActionButton(
67     child: Icon(Icons.add),
68     onPressed: () {
69       // + 버튼 클릭시 버킷 생성 페이지로 이동

```



ListTile 위젯을 활용하면 박스 안에 여러 영역에 다른 위젯을 손쉽게 배치할 수 있습니다.

```
ListTile(  
  leading: icon,  
  title: "Widget of the week",  
  subtitle: "#54 ...",  
  trailing: icon,  
);
```



Widget of the week  
#54



leading

title  
subtitle

trailing

좀 더 상세한 설명은 아래 공식 문서를 참고해 주세요.

▼ **[코드스니펫] ListTile 공식문서 URL**

<https://api.flutter.dev/flutter/material/ListTile-class.html>

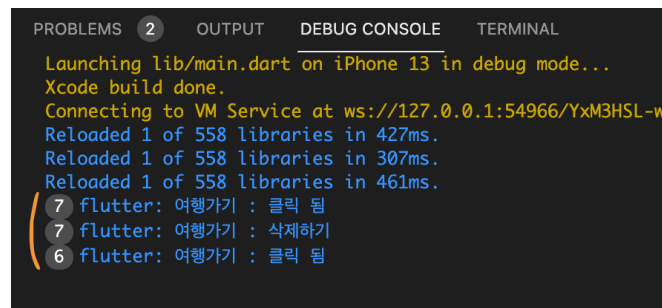
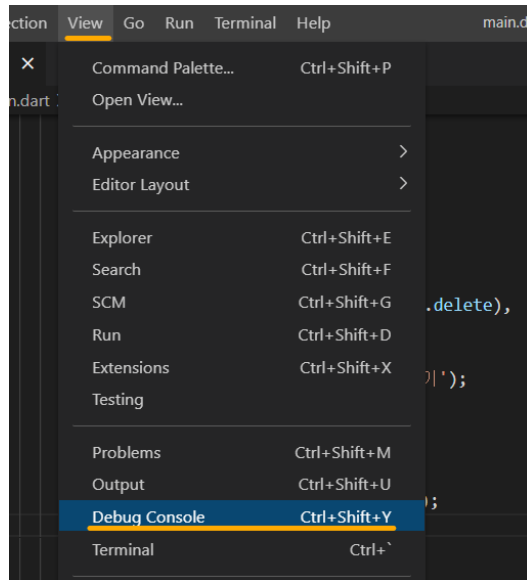
저장해주시면 예뮬레이터에 **여행가기** 항목이 추가된 것을 볼 수 있습니다.

버킷 리스트

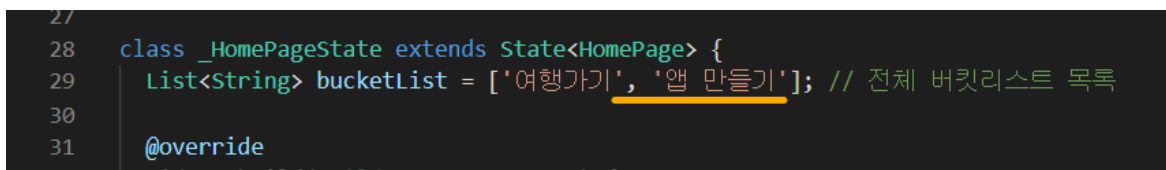
여행가기



클릭시 print문은 **view** → **Debug Console** 을 선택하셔서 보실 수 있습니다.



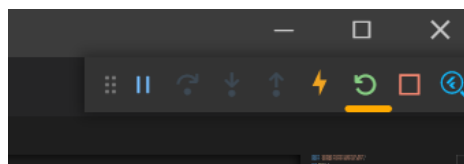
5. `bucketList` 상태 변수에 `'앱 만들기'` 를 추가해 봅시다.



저장을 해도 에뮬레이터에 반영이 안되실 겁니다.

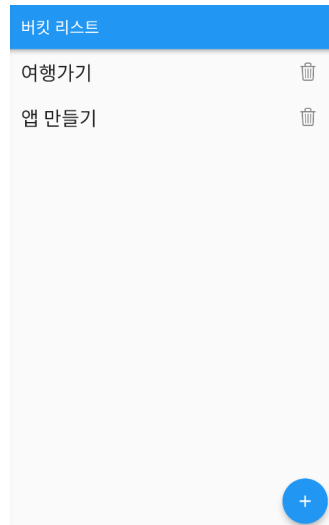
StatefulWidget의 상태 값을 코드상에서 변경하는 경우 `Restart` 를 해야 앱에 반영됩니다.

우측 상단에 `Restart` 버튼을 눌러주세요.



앱에 버킷 리스트가 보이는 것을 보실 수 있습니다.





여기까지 버킷 리스트 조회 기능 완성!

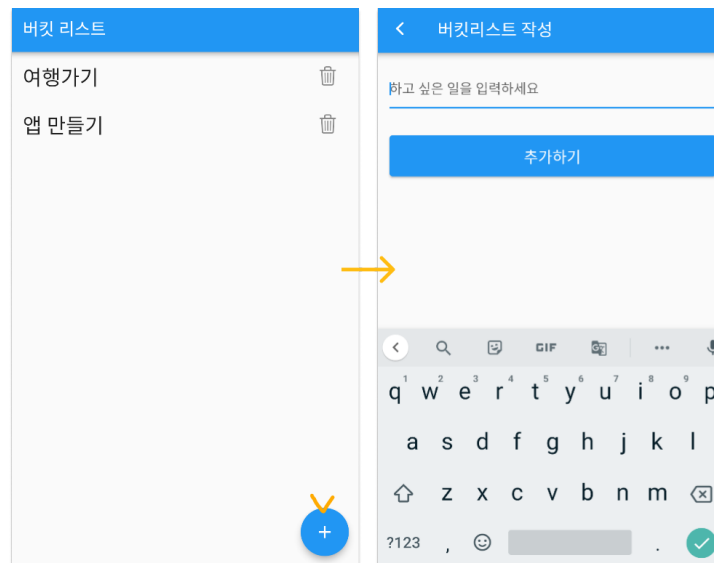
### ▼ 3) 버킷 리스트 작성(Create) : 유효성 검사



#### 요구사항

- 추가하기 버튼을 눌렀을 때 텍스트를 입력하지 않은 경우, 경고 메시지를 보여주기

1. `FloatingActionButton` 을 클릭하여 `CreatePage` 로 이동해 주세요.

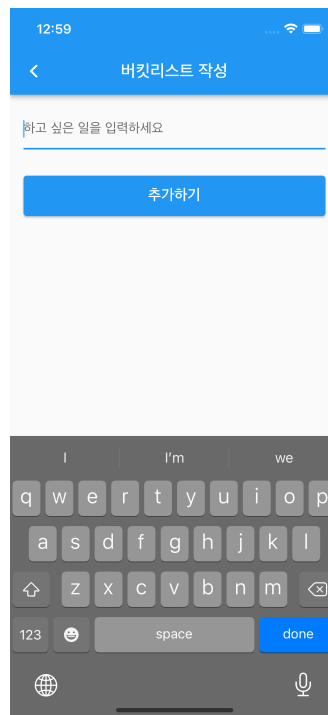
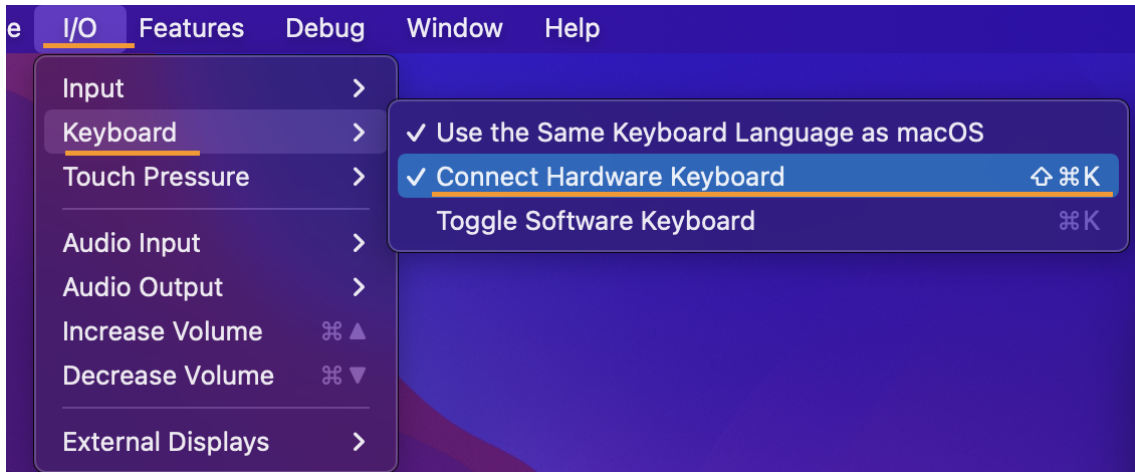


### ▼ iOS 에뮬레이터에서 키보드를 보이게 하는 방법



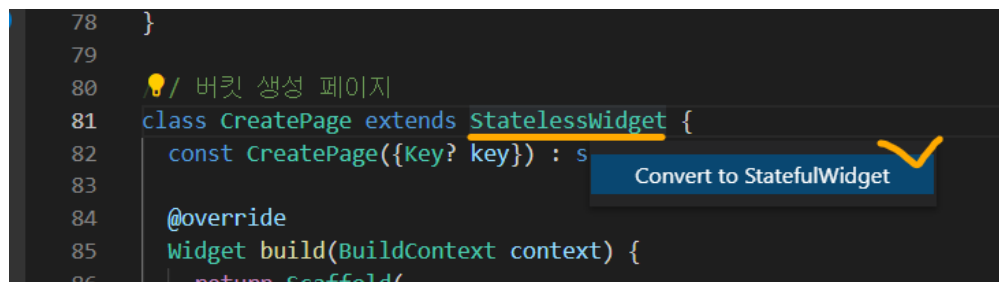
iOS 에뮬레이터 선택 → I/O → Keyboard → Connect Hardware Keyboard 를 해제하면 키보드가 올라옵니다.

또는 iOS 에뮬레이터 선택 후 단축키로 `Cmd + Shift + K`



2. 텍스트 입력 여부라는 상태에 따라 화면을 갱신하여 에러 메시지를 띄워야 하므로 `CreatePage` 를 `StatefulWidget` 으로 변경해 줍니다.

81번째 라인에 `StatelessWidget` 을 클릭한 뒤 Quick Fix(`Ctrl/Cmd + .`)를 누르고 `Convert to StatefulWidget` 을 선택해주세요.



`CreatePage` 의 상태를 관리하는 `_CreatePageState` 클래스가 추가 되었습니다.

```

79
80 /// 버킷 생성 페이지
81 class CreatePage extends StatefulWidget {
82   const CreatePage({Key? key}) : super(key: key);
83
84   @override
85   State<CreatePage> createState() => _CreatePageState();
86 }
87
88 class _CreatePageState extends State<CreatePage> {
89   @override
90   Widget build(BuildContext context) {
91     return Scaffold(
92       appBar: AppBar(

```



**추가하기** 버튼을 클릭하는 경우, 현재 작성한 텍스트 값을 가져와 봅시다.

3. 먼저 `TextField`의 값을 가져올 수 있도록 `textController`를 만들어줍니다. 아래 코드스니펫을 복사해 88번째 라인 뒤에 붙여 넣어 주세요.

▼ [코드스니펫] CreatePage / textController 생성

```

// TextField의 값을 가져올 때 사용합니다.
TextEditingController textController = TextEditingController();

```

```

87
88 class _CreatePageState extends State<CreatePage> {
89   // TextField의 값을 가져올 때 사용합니다.
90   TextEditingController textController = TextEditingController();
91
92   @override
93   Widget build(BuildContext context) {
94     return Scaffold(
95       appBar: AppBar(

```

4. `textController`를 `TextField` 위젯과 연결해 봅시다. 코드스니펫을 복사해 110번째 줄 맨 뒤에 붙여 넣어주세요.

▼ [코드스니펫] CreatePage / textController 연결

```

controller: textController, // 연결해 줍니다.

```

```

106 padding: const EdgeInsets.all(16),
107   child: Column(
108     children: [
109       // 텍스트 입력창
110       TextField(
111         controller: textController, // 연결해 줍니다.
112         autofocus: true,
113         decoration: InputDecoration(
114           hintText: "하고 싶은 일을 입력하세요",

```

5. **추가하기** 버튼을 누를 때, `textController` 를 이용하여 `TextField` 에 입력된 가져와 보도록 하겠습니다. 코드스니펫을 복사해 132번째 라인 맨 뒤에 추가해 주세요.

▼ [코드스니펫] CreatePage / TextField 값 가져오기

```

String job = textController.text; // 값 가져오기
print(job);

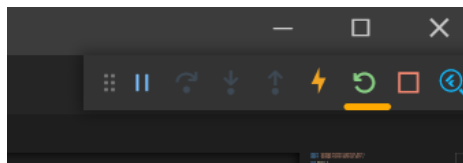
```

```

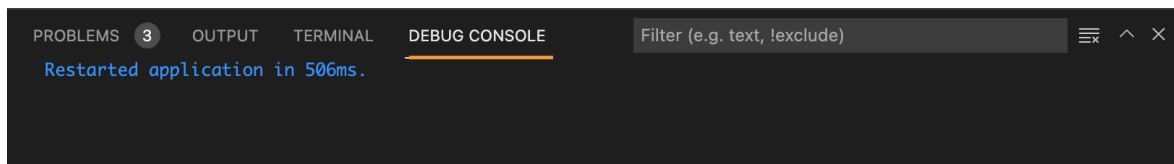
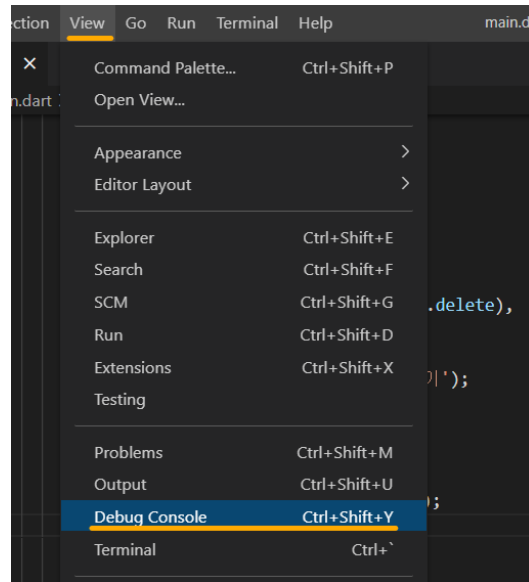
123   child: Text(
124     "추가하기",
125     style: TextStyle(
126       fontSize: 18,
127     ), // TextStyle
128   ), // Text
129   onPressed: () {
130     // 추가하기 버튼 클릭시
131     String job = textController.text; // 값 가져오기
132     print(job);
133   },
134   ), // ElevatedButton
135   ), // SizedBox
136

```

저장한 뒤 **Restart** 버튼을 눌러주세요.

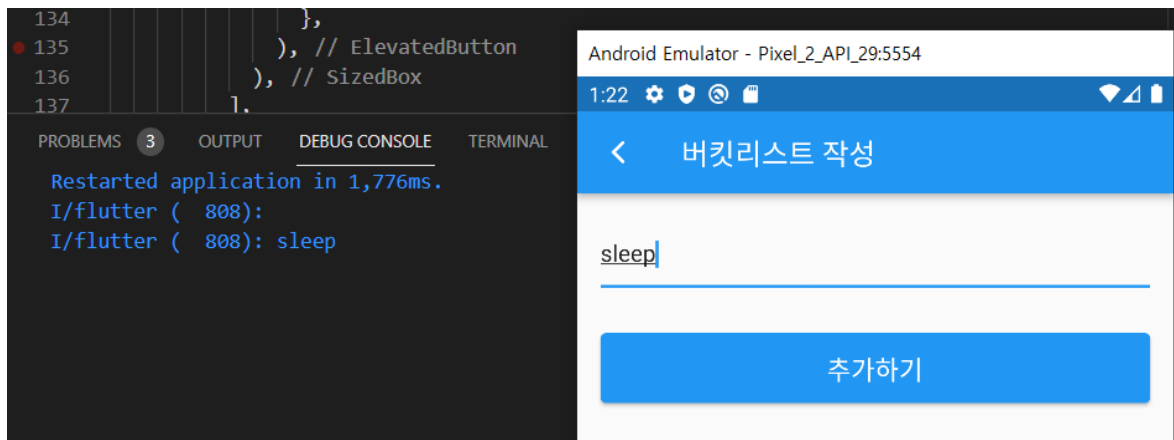


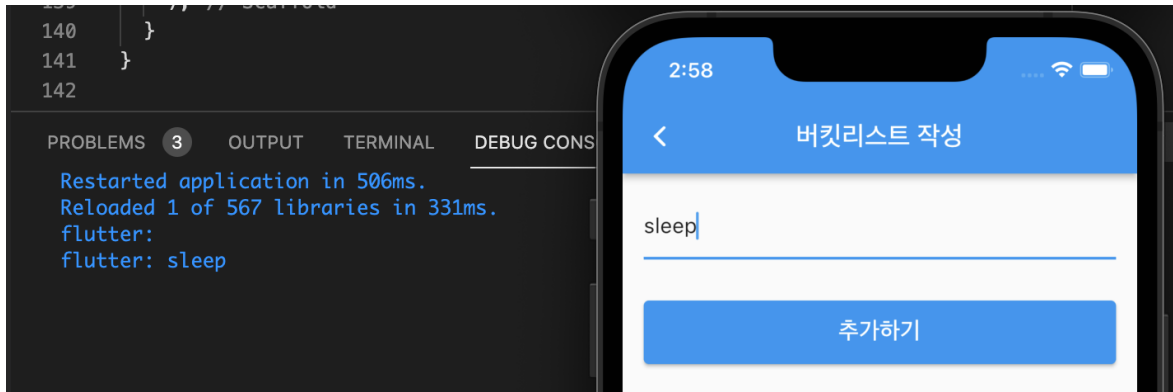
**view** → **Debug Console** 을 선택하여 콘솔 창을 띄워주세요.



에뮬레이터에 텍스트를 입력한 뒤 **추가하기** 버튼을 누르면 콘솔창에 해당 값이 잘 가져와지는 것을 확인하실 수 있습니다.

💡 Android 에뮬레이터에 한글 자판이 설치되어 있지 않아 영어로만 작성이 될 거예요!



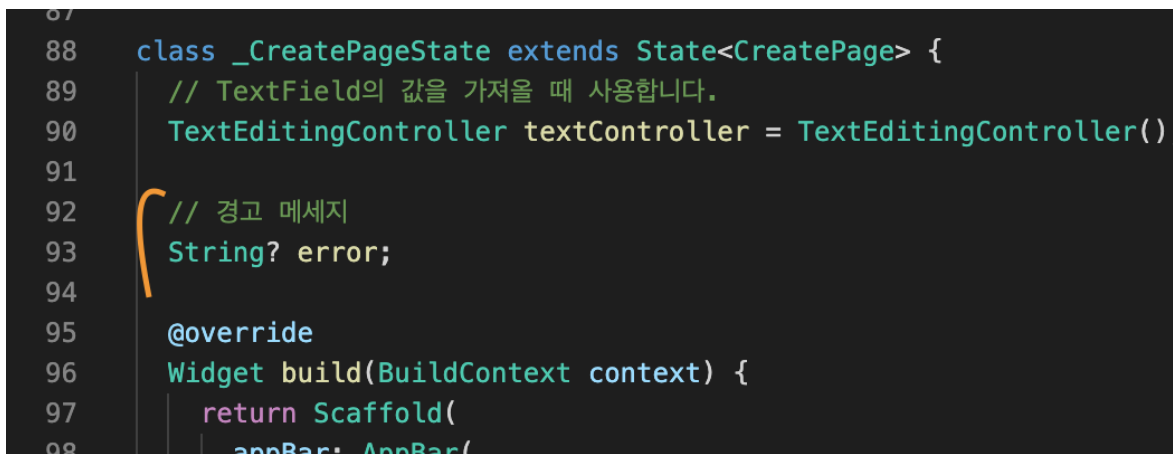


💡 이제 `job` 변수가 비어있는 경우, 경고 메시지를 띄워보도록 하겠습니다.

6. 먼저 경고 메시지를 담는 변수 `error` 를 만들어 보겠습니다.  
코드스니펫을 복사해서 91번째 라인 뒤에 붙여 넣어 주세요.

▼ [코드스니펫] CreatePage / error

```
// 경고 메시지
String? error;
```



7. `TextField` 의 `errorText` 에 `error` 변수를 할당해 줍니다. 코드스니펫을 복사해 117번째 라인 뒤에 붙여 주세요.

▼ [코드스니펫] CreatePage / ErrorText

```
errorText: error,
```

```

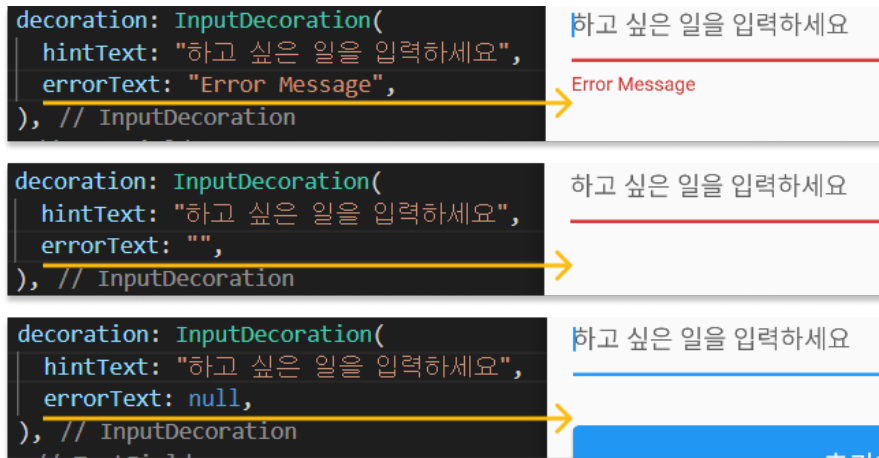
109         padding: const EdgeInsets.all(10),
110         child: Column(
111             children: [
112                 // 텍스트 입력창
113                 TextField(
114                     controller: textController, // 연결해 줍니다.
115                     autofocus: true,
116                     decoration: InputDecoration(
117                         hintText: "하고 싶은 일을 입력하세요",
118                         errorText: error,
119                     ), // InputDecoration
120                 ), // TextField
121                 SizedBox(height: 32),
122                 // 추가하기 버튼
123                 SizedBox(

```



`TextField`에 `errorText`라는 파라미터에 문구를 넣으면 아래 이미지와 같이 경고 문구를 넣을 수 있습니다.

경고 문구를 안 띄우려면 `null`로 할당해야 하므로 `error`를 `String?` 타입으로 변수를 만들었습니다.



8. 이제 `추가하기` 버튼을 누를 때, 텍스트가 비어있는 경우 에러 메시지를 띄워 보시다.

코드스니펫을 복사해 136번째 라인을 지우고 붙여 넣어 주세요.

#### ▼ [코드스니펫] CreatePage / validation

```

if (job.isEmpty) {
    setState(() {
        error = "내용을 입력해주세요."; // 내용이 없는 경우 에러 메시지
    });
} else {
    setState(() {
        error = null; // 내용이 있는 경우 에러 메시지 숨기기
    });
}

```



**error** 변수의 값이 바뀌는 경우 화면을 갱신해줘야 하므로 `setState()` 함수로 감싸 주었습니다.

```

130         fontSize: 18,
131       ), // TextStyle
132     ), // Text
133     onPressed: () {
134       // 추가하기 버튼 클릭시
135       String job = textController.text; // 값 가져오기
136       if (job.isEmpty) {
137         setState(() {
138           error = "내용을 입력해주세요."; // 내용이 없는 경우 에러 메시지
139         });
140       } else {
141         setState(() {
142           error = null; // 내용이 있는 경우 에러 메시지 숨기기
143         });
144       }
145     },
146   ), // ElevatedButton
147 ), // SizedBox

```

저장 후 텍스트가 없는 상태에서 추가하기 버튼을 누르면 에러 메시지가 뜹니다.

텍스트가 있다면 버튼을 눌러도 에러 메시지가 뜨지 않는 것을 볼 수 있습니다.

▼ 4) 버킷 리스트 작성(Create) : 화면 간 데이터 전달





#### 요구사항

- 텍스트를 입력 후 추가하기 버튼 클릭 시, 홈 화면으로 이동하며 버킷 리스트에 항목이 추가된다.

1. `CreatePage` 에서 입력한 텍스트를 `HomePage` 에 있는 `bucketList` 에 추가해야합니다.

코드스니펫을 복사해 143번째 라인 뒤에 붙여 넣어 주세요.

#### ▼ [코드스니펫] `CreatePage` / `pop`

```
Navigator.pop(context, job); // job 변수를 반환하며 화면을 종료합니다.
```



`Navigator.pop()` 호출시 이전 페이지에 전달할 파라미터를 넣어줄 수 있습니다.

```

133         onPressed: () {
134             // 추가하기 버튼 클릭시
135             String job = textController.text; // 값 가져오기
136             if (job.isEmpty) {
137                 setState(() {
138                     error = "내용을 입력해주세요."; // 내용이 없는 경우 에러 메시지
139                 });
140             } else {
141                 setState(() {
142                     error = null; // 내용이 있는 경우 에러 메시지 숨기기
143                 });
144                 Navigator.pop(context, job); // job 변수를 반환하며 화면을 종료합니다.
145             }
146         },

```

2. `CreatePage` 에서 반환한 값을 `HomePage` 에서 받아봅시다. 68번째 라인에 `onPressed` 함수를 이미지와 같이 수정해 봅시다.

- 68번째 라인 : `async` 추가
- 70번째 라인 : `String? job = await` 추가
- 74번째 라인 : `print(job);` 추가

```


65         ), // ListView.builder
66         floatingActionButton: FloatingActionButton(
+ 67             child: Icon(Icons.add),
68             onPressed: () async {
69                 // + 버튼 클릭시 버킷 생성 페이지로 이동
70                 String? job = await Navigator.push(
71                     context,
72                     MaterialPageRoute(builder: (_) => CreatePage()),
73                 );
74                 print(job);
75             },
76         ), // FloatingActionButton
77     ); // Scaffold

```

 **async** 와 **await** 은 무엇인가요?

**await** 은 `Navigator.push()` 로 화면을 띄운 뒤, 해당 화면이 종료될 때까지 70번째 라인에서 기다리도록 만들어주는 코드입니다. 이후 화면이 종료되면 `job` 이라는 변수에 반환 된 파라미터를 할당하고 다음 74번째 라인이 진행됩니다.

**await** 을 사용하려면 해당 함수에 **async** 키워드를 넣어야 합니다.

 `CreatePage` 에 이동한 뒤 AppBar의 뒤로가기 버튼을 눌러서 `HomePage` 로 되돌아 오는 경우 반환하는 `null` 을 반환하기 때문에 그래서 `String?` 로 타입을 지정 하였습니다.

```

66         floatingActionButton: FloatingActionButton(
67             child: Icon(Icons.add),
68             onPressed: () async {
69                 // + 버튼 클릭시 버킷 생성 페이지로 이동
70                 String? job = await Navigator.push(
71                     context,
72                     MaterialPageRoute(builder: (_) => CreatePage()),
73                 );
74                 print(job);
75             },
76         ), // FloatingActionButton

```

3. **Restart** 를 한 뒤 정상적으로 작동하는지 테스트해 줍니다.

`CreatePage` 에서 `sleep` 이라고 작성한 뒤 **추가하기** 버튼을 누르면 콘솔창에 아래와 같이 파라미터가 받아지는 것을 볼 수 있습니다.

```

PROBLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL
Restarted application in 2,177ms.
W/IInputConnectionWrapper( 808): getTextBeforeCursor on inactive InputConnection
W/IInputConnectionWrapper( 808): getSelectedText on inactive InputConnection
W/IInputConnectionWrapper( 808): getTextAfterCursor on inactive InputConnection
I/flutter ( 808): sleep

```

반면, `CreatePage`에서 AppBar의 뒤로가기 버튼을 눌러 되돌아 오는 경우 `null`이라고 반환 되는 것을 볼 수 있습니다.

```
PROBLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL
Restarted application in 1,819ms.
I/flutter ( 808): null
```

4. 전달 받은 값을 `bucketList` 변수에 추가해 봅시다. 74번째 print 구문을 코드스니펫으로 교체해 주세요.

▼ [코드스니펫] HomoPage / bucketList 추가

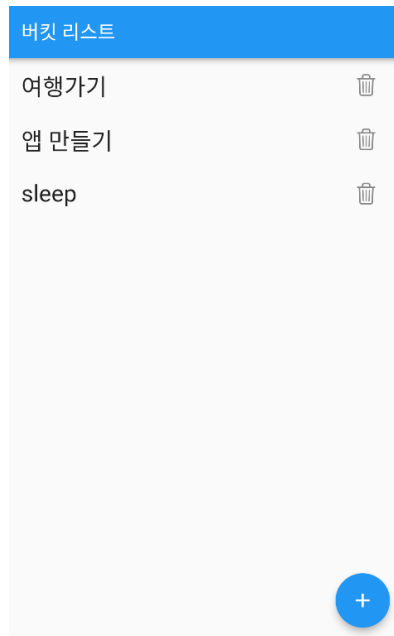
```
if (job != null) {
  setState(() {
    bucketList.add(job); // 버킷 리스트에 추가
  });
}
```

💡 `job`이 `null`이 아닌 경우 `bucketList`에 추가하도록 `if` 문으로 감싸주었습니다.

💡 `bucketList`에 전달 받은 `job`을 추가한 뒤 화면을 갱신하기 위해 `setState()`로 감싸주었습니다.

```
68      onPressed: () async {
69        // + 버튼 클릭시 버킷 생성 페이지로 이동
70        String? job = await Navigator.push(
71          context,
72          MaterialPageRoute(builder: (_) => CreatePage()),
73        );
74        if (job != null) {
75          setState(() {
76            bucketList.add(job); // 버킷 리스트에 추가
77          });
78        }
79      },
80    ), // FloatingActionButton
81  ); // Scaffold
```

5. 저장한 뒤 버킷 리스트를 작성해 보면, 홈 화면도 추가되는 것을 보실 수 있습니다.



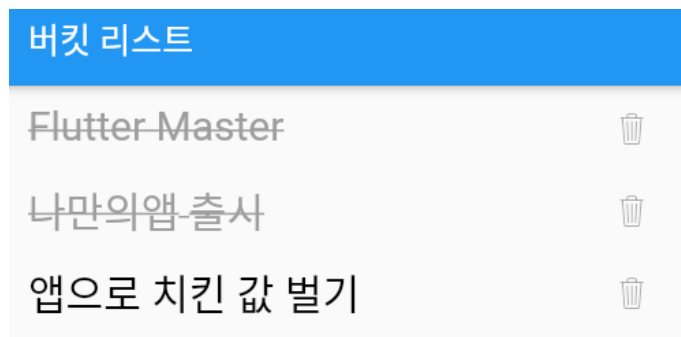
#### ▼ 5) 버킷 리스트 수정(Update)



#### 요구사항

- 버킷 리스트 아이템을 클릭하여 달성 여부 표시하기

달성된 항목은 폰트 색상을 회색으로 변경하고, 중앙에 선을 그어서 완료 표시를 해보겠습니다.



1. 버킷 리스트 항목에 할 일에 해당하는 `String` 뿐만 아니라 `완료 여부` 를 나타내는 상태를 추가해야 합니다. `Bucket` 클래스를 만들어 문제를 해결해 봅시다.

19번째 줄에 코드스니펫을 붙여 넣어 주세요.

#### ▼ [코드스니펫] Bucket class

```
/// 버킷 클래스
class Bucket {
  String job; // 할 일
  bool isDone; // 완료 여부

  Bucket(this.job, this.isDone); // 생성자
}
```

```

17 }
18 }
19
20 /// 버킷 클래스
21 class Bucket {
22   String job; // 할 일
23   bool isDone; // 완료 여부
24
25   Bucket(this.job, this.isDone); // 생성자
26 }
27
28 /// 홈 페이지
29 class HomePage extends StatefulWidget {
30   const HomePage({Key? key}) : super(key: key);
31

```

2. `bucketList` 변수의 타입을 `List<Bucket>` 으로 변경하고 빈 배열로 만들어줍니다.

코드스니펫을 복사해 37번째 라인을 교체해 주세요.

▼ [코드스니펫] HomePage / BucketList 타입 변경

```
List<Bucket> bucketList = []; // 전체 버킷리스트 목록
```

```

35
36 class _HomePageState extends State<HomePage> {
37   List<Bucket> bucketList = []; // 전체 버킷리스트 목록
38
39   @override
40   Widget build(BuildContext context) {

```

3. 50번째 라인에 `bucket` 의 타입을 `String` 에서 `Bucket` 으로 변경해 줍니다.

💡 `bucketList` 에 타입이 `List<Bucket>` 이므로 `bucketList[index]` 의 타입은 `Bucket` 이 됩니다.

```

46 ? Center(child: Text("버킷 리스트를 작성해 주세요."))
47 : ListView.builder(
48   itemCount: bucketList.length, // bucketList 개수 만큼 보여주기
49   itemBuilder: (context, index) {
50     Bucket bucket = bucketList[index]; // index에 해당하는 bucket 가져오기
51     return ListTile(
52       // 버킷 리스트 할 일
53       title: Text(
54         bucket

```

4. 54번째 라인에 `bucket.job` 으로 변경해주세요.

💡 `bucket` 의 할 일은 `job` 이라는 속성에 들어있습니다.

```

50     Bucket bucket = bucketList[index];
51     return ListTile(
52       // 버킷 리스트 할 일
53       title: Text(
54         bucket.job,
55         style: TextStyle(
56           fontSize: 24,
57         ), // TextStyle

```

5. 마지막으로 84번째 라인을 아래 코드스니펫으로 변경해주세요.

▼ [코드스니펫] HomePage / Bucket 추가

```

Bucket newBucket = Bucket(job, false);
bucketList.add(newBucket); // 버킷 리스트에 추가

```



CreatePage 로 받은 job 을 Bucket 클래스의 생성자에 전달해서 Bucket 인스턴스를 BucketList 에 추가해 줍니다.

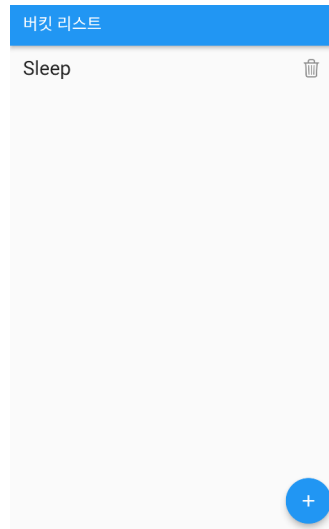
완료 여부를 나타내는 isDone 은 기본 값을 false 로 넣어주었습니다.

```

79     context,
80     MaterialPageRoute(builder: (_) => CreatePage()),
81   );
82   if (job != null) {
83     setState(() {
84       Bucket newBucket = Bucket(job, false);
85       bucketList.add(newBucket); // 버킷 리스트에 추가
86     });
87   }
88 },
89 ), // FloatingActionButton
90 ); // Scaffold

```

6. 저장을 하면 에뮬레이터에 에러가 발생하는데, Restart 를 해주시면 정상적으로 작동 됩니다. FloatingActionButton 을 누르고 버킷 리스트를 추가해 보세요.



7. 이제 버킷 리스트 아이템 클릭시 `Bucket` 클래스의 `isDone` bool 값을 반전시켜 보도록 하겠습니다.

코드스니펫을 복사해서 69번째 라인을 변경해 주세요.

▼ [코드스니펫] HomePage / isDone 상태 변경

```
setState(() {
    bucket.isDone = !bucket.isDone;
});
```



`bucket`의 `isDone` 상태에 따라 화면에 다르게 보여줄 계획이므로, `isDone` 변경시 화면을 갱신하기 위해 `setState()`로 감쌌습니다.

```
63 // 삭제 버튼 클릭시
64 print('$bucket : 삭제하기');
65 },
66 ), // IconButton
67 onTap: () {
68 // 아이템 클릭시
69 setState(() {
70 bucket.isDone = !bucket.isDone; // isDone 상태 변경
71 });
72 },
73 ); // ListTile
74 },
75 ), // ListView.builder
76 floatingActionButton: FloatingActionButton(
77 child: Icon(Icons.add),
```

8. `isDone`의 값에 따라 Text 위젯을 다르게 보여주도록 하겠습니다.

코드스니펫을 복사해 56번째 라인 뒤에 붙여 넣어주세요.

▼ [코드스니펫] HomePage / Text Decoration

```
color: bucket.isDone ? Colors.grey : Colors.black,
```

```

decoration: bucket.isDone
? TextDecoration.lineThrough
: TextDecoration.none,

```

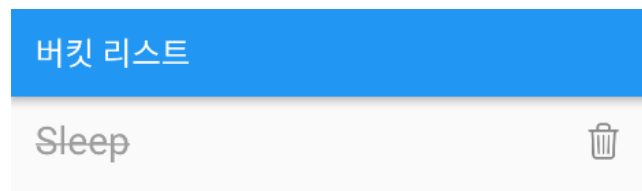
```

51      return ListTile(
52        // 버킷 리스트 할 일
53        title: Text(
54          bucket.job,
55          style: TextStyle(
56            fontSize: 24,
57            color: bucket.isDone ? Colors.grey : Colors.black,
58            decoration: bucket.isDone
59              ? TextDecoration.lineThrough
60              : TextDecoration.none,
61          ), // TextStyle
62        ), // Text
63        // 삭제 아이콘 버튼

```

 `decoration` 속성에 `TextDecoration.lineThrough`를 넣어주면 텍스트 중앙에 선을 그을 수 있습니다.

9. 저장해 주신 뒤, 에뮬레이터에 버킷 리스트 아이템을 클릭해 보면 아래와 같이 정상 작동하는 것을 보실 수 있습니다.



#### ▼ 6) 버킷 리스트 삭제(Delete)

##### 요구사항

- 삭제 아이콘을 누르는 경우, 삭제 확인 팝업(dialog)이 뜨고 **확인** 버튼을 누르는 경우 아이템이 삭제됩니다.

1. 삭제 버튼 클릭 이벤트는 66번째 `onPressed`의 함수로 받을 수 있습니다.

```

63      // 삭제 아이콘 버튼
64      trailing: IconButton(
65        icon: Icon(CupertinoIcons.delete),
66        onPressed: () {
67          // 삭제 버튼 클릭시
68          print('$bucket : 삭제하기');
69        },
70      ), // IconButton

```

먼저 삭제 버튼을 누르는 경우 `Dialog`를 띄워보도록 하겠습니다.

코드스니펫을 복사해 68번째 라인을 변경해주세요.

▼ [코드스니펫] `HomePage / showDialog`



```

showDialog(
  context: context,
  builder: (context) {
    return AlertDialog(
      title: Text("정말로 삭제하시겠습니까?"),
    );
  },
);

```


```

62      ), // Text
63      // 삭제 아이콘 버튼
64      trailing: IconButton(
65        icon: Icon(CupertinoIcons.delete),
66        onPressed: () {
67          // 삭제 버튼 클릭시
68          showDialog(
69            context: context,
70            builder: (context) {
71              return AlertDialog(
72                title: Text("정말로 삭제하시겠습니까?"),
73              ); // AlertDialog
74            },
75          );
76        },
77      ), // IconButton
78      onTap: () {

```

삭제 버튼을 눌러서 `Dialog` 를 확인해보세요



 Dialog 배경을 클릭하여 끌 수 있습니다.

2. `AlertDialog` 에 취소 와 확인 버튼을 추가해 보도록 하겠습니다.

코드스니펫을 복사해 72번째 라인 뒤에 붙여 넣어주세요.

▼ [코드스니펫] HomePage / Dialog actions

```
actions: [
  // 취소 버튼
  TextButton(
    onPressed: () {
      Navigator.pop(context);
    },
    child: Text("취소"),
  ),
  // 확인 버튼
  TextButton(
    onPressed: () {
      Navigator.pop(context);
    },
    child: Text(
      "확인",
      style: TextStyle(color: Colors.pink),
    ),
  ),
],
```

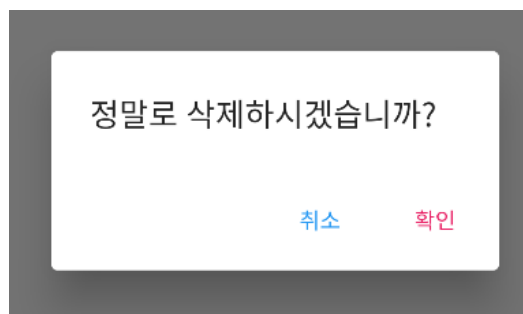


`AlertDialog`의 `actions` 는 배열로 원하는 위젯을 넣을 수 있는 파라미터 입니다.



버튼을 클릭하는 경우 `Navigator.pop(context);` 를 호출하여 Dialog를 종료할 수 있습니다.

```
lib > main.dart > _HomePageState > build
// 삭제 버튼 클릭 시
64 trailing: IconButton(
65   icon: Icon(CupertinoIcons.delete),
66   onPressed: () {
67     // 삭제 버튼 클릭 시
68     showDialog(
69       context: context,
70       builder: (context) {
71         return AlertDialog(
72           title: Text("정말로 삭제하시겠습니까?"),
73           actions: [
74             // 취소 버튼
75             TextButton(
76               onPressed: () {
77                 Navigator.pop(context);
78               },
79               child: Text("취소"),
80             ), // TextButton
81             // 확인 버튼
82             TextButton(
83               onPressed: () {
84                 Navigator.pop(context);
85               },
86               child: Text(
87                 "확인",
88                 style: TextStyle(color: Colors.pink),
89               ), // Text
90             ), // TextButton
91           ],
92         ); // AlertDialog
93       },
94     );
95   },
  }
```



3. **확인** 버튼 클릭 시 `BuckList` 에서 항목을 삭제해 보도록 하겠습니다.


코드스니펫을 복사해 83번째 뒤에 붙여 넣어주세요.

▼ [코드스니펫] `HomePage / delete bucket`

```

setState(() {
  // index에 해당하는 항목 삭제
  bucketList.removeAt(index);
});

```

 `List.removeAt(index)` 를 이용하면 배열에서 index에 해당하는 항목을 삭제할 수 있습니다.

💡 아이템을 삭제한 뒤 화면을 갱신해야 하므로 `setState()` 로 감싸 주었습니다.

```

80     }, // TextButton
81     // 확인 버튼
82     TextButton(
83       onPressed: () {
84         setState(() {
85           // index에 해당하는 항목 삭제
86           bucketList.removeAt(index);
87         });
88         Navigator.pop(context);
89       },
90       child: Text(
91         "확인"

```

4. 저장 후 다이얼로그를 다시 띄우고 **확인** 버튼을 누르면 버킷 리스트에서 선택한 항목이 사라지는 것을 확인하실 수 있습니다.



▼ 7) 리팩토리

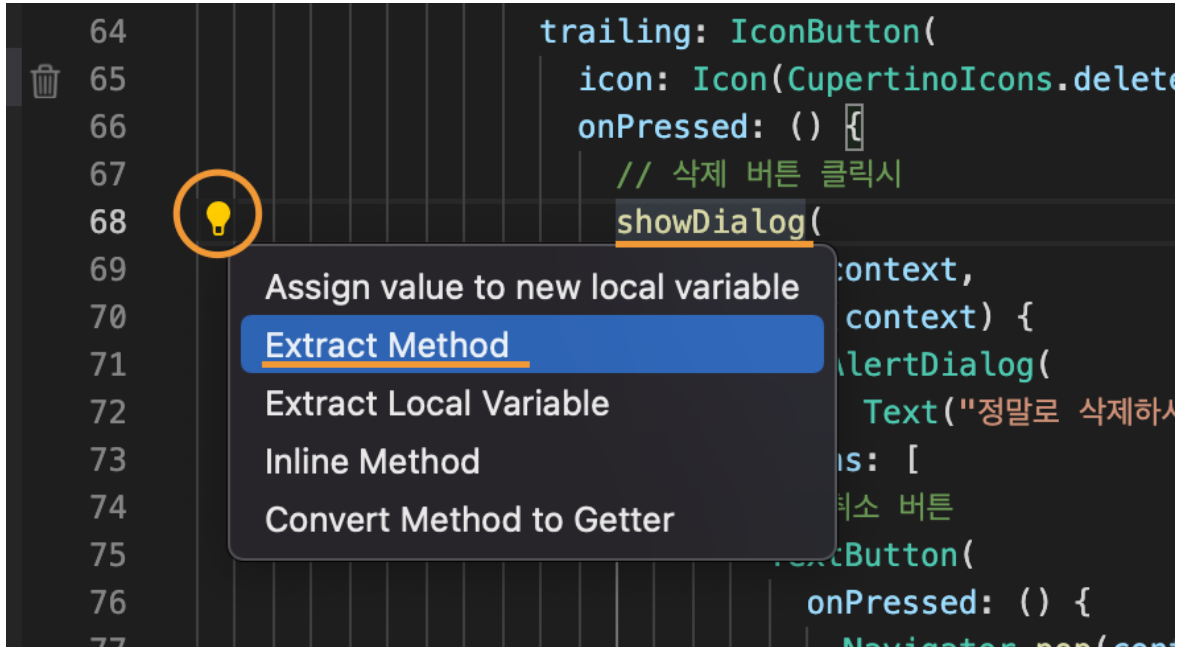
💡 Dialog를 띄우는 로직()이 너무 길어서 보기 힘든 것 같습니다. 해당 로직을 별도의 함수로 분리 시켜보도록 하겠습니다.

💡 이와 같이 동작의 변경 없이 코드를 정리하는 과정을 리팩토리(Refactory)라고 부릅니다.

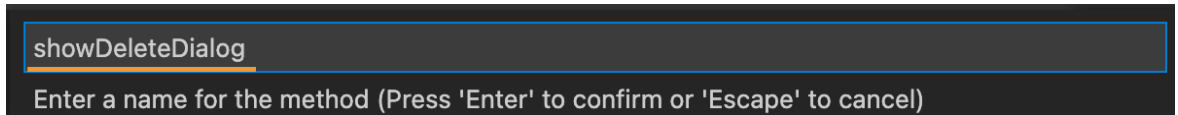
1. `showDeleteDialog` 메소드(함수)로 분리하기

💡 코드를 함수로 나누면 보기도 좋고 재활용도 할 수 있습니다.

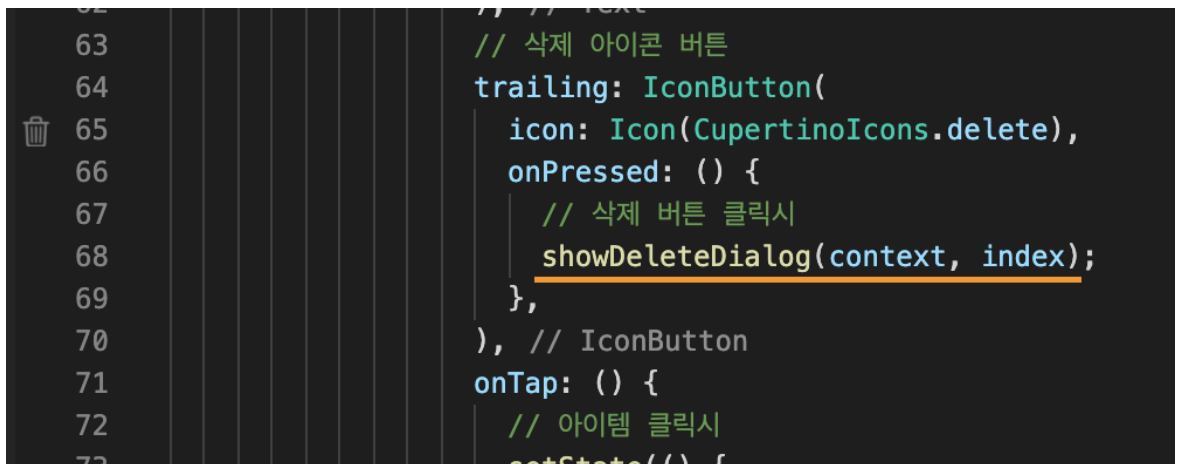
68번째 줄에 `showDialog`를 클릭한 뒤 왼쪽에 전구(💡) 아이콘을 선택하고, `Extract Method`를 선택해 주세요.



메소드(함수)의 이름을 입력하라고 뜨면 `showDeleteDialog`라고 입력한 뒤 엔터를 누르고 저장(`Ctrl/Cmd + S`)을 눌러주세요.



그러면 아래와 같이 68번째 줄에 함수를 호출하도록 바꿉니다.



그리고 99번째 라인으로 내려가보면 `showDeleteDialog`라는 함수가 생성되어 있는 것을 볼 수 있습니다.

```

99   Future<dynamic> showDeleteDialog(BuildContext context, int index) {
100   return showDialog(
101     context: context,
102     builder: (context) {
103       return AlertDialog(
104         title: Text("정말로 삭제하시겠습니까?"),
105         actions: [
106           // 취소 버튼
107           TextButton(
108             onPressed: () {
109               Navigator.pop(context);
110             },
111             child: Text("취소"),
112           ), // TextButton
113           // 확인 버튼
114           TextButton(
115             onPressed: () {
116               setState(() {
117                 // 삭제
118                 bucketList.removeAt(index);
119               });
120               Navigator.pop(context);
121             },
122             child: Text(
123               "확인",
124               style: TextStyle(color: Colors.pink),
125             ), // Text
126           ), // TextButton
127         ],
128       ); // AlertDialog
129     },
130   );
131 }

```

2. 99번째 줄을 보면 함수의 반환 타입이 `Future<dynamic>` 이라고 되어있는데, 이 부분을 `void` 로 변경하고, 100번째 줄에 있는 `return` 을 삭제해주세요.

```

98
99   void showDeleteDialog(BuildContext context, int index) {
100     showDialog(
101       context: context,
102       builder: (context) {
103         return AlertDialog(
104           title: Text("정말로 삭제하시겠습니까?"),
105           actions: [

```



`void` 는 함수가 아무것도 반환하지 않는다는 의미이고, 이에 맞추어 `return` 을 삭제했습니다.

3. 저장한 뒤, 버킷을 하나 만들고 삭제해보면 정상적으로 작동하는 것을 확인할 수 있습니다.

▼ `main.dart` 완성 코드

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

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

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      home: HomePage(),
    );
  }
}

/// 버킷 클래스
class Bucket {
  String job; // 할 일
  bool isDone; // 완료 여부

  Bucket(this.job, this.isDone); // 생성자
}

/// 홈 페이지
class HomePage extends StatefulWidget {
  const HomePage({Key? key}) : super(key: key);

  @override
  State<HomePage> createState() => _HomePageState();
}

class _HomePageState extends State<HomePage> {
  List<Bucket> bucketList = []; // 전체 버킷리스트 목록

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("버킷 리스트"),
      ),
      body: bucketList.isEmpty
        ? Center(child: Text("버킷 리스트를 작성해 주세요."))
        : ListView.builder(
            itemCount: bucketList.length, // bucketList 개수 만큼 보여주기
            itemBuilder: (context, index) {
              Bucket bucket = bucketList[index]; // index에 해당하는 bucket 가져오기
              return ListTile(
                // 버킷 리스트 할 일
                title: Text(
                  bucket.job,
                  style: TextStyle(
                    fontSize: 24,
                    color: bucket.isDone ? Colors.grey : Colors.black,
                    decoration: bucket.isDone
                      ? TextDecoration.lineThrough
                      : TextDecoration.none,
                  ),
                ),
                // 삭제 아이콘 버튼
                trailing: IconButton(
                  icon: Icon(CupertinoIcons.delete),
                  onPressed: () {
                    // 삭제 버튼 클릭시
                    showDeleteDialog(context, index);
                  },
                ),
                onTap: () {
                  // 아이템 클릭시
                  setState(() {

```

```

        bucket.isDone = !bucket.isDone;
      });
    },
  );
},
),
floatingActionButton: FloatingActionButton(
  child: Icon(Icons.add),
  onPressed: () async {
    // + 버튼 클릭시 버킷 생성 페이지로 이동
    String? job = await Navigator.push(
      context,
      MaterialPageRoute(builder: (_) => CreatePage()),
    );
    if (job != null) {
      setState(() {
        Bucket newBucket = Bucket(job, false);
        bucketList.add(newBucket); // 버킷 리스트에 추가
      });
    }
  },
),
);
}

void showDeleteDialog(BuildContext context, int index) {
  showDialog(
    context: context,
    builder: (context) {
      return AlertDialog(
        title: Text("정말로 삭제하시겠습니까?"),
        actions: [
          // 취소 버튼
          TextButton(
            onPressed: () {
              Navigator.pop(context);
            },
            child: Text("취소"),
          ),
          // 확인 버튼
          TextButton(
            onPressed: () {
              setState(() {
                // 삭제
                bucketList.removeAt(index);
              });
              Navigator.pop(context);
            },
            child: Text(
              "확인",
              style: TextStyle(color: Colors.pink),
            ),
          ),
        ],
      );
    },
  );
}

/// 버킷 생성 페이지
class CreatePage extends StatefulWidget {
  const CreatePage({Key? key}) : super(key: key);

  @override
  State<CreatePage> createState() => _CreatePageState();
}

class _CreatePageState extends State<CreatePage> {
  // TextField의 값을 가져올 때 사용합니다.
  TextEditingController textController = TextEditingController();

  // 경고 메시지
  String? error;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(

```



```

        title: Text("버킷리스트 작성"),
        // 뒤로가기 버튼
        leading: IconButton(
          icon: Icon(CupertinoIcons.chevron_back),
          onPressed: () {
            Navigator.pop(context);
          },
        ),
      ),
      body: Padding(
        padding: const EdgeInsets.all(16),
        child: Column(
          children: [
            // 텍스트 입력창
            TextField(
              controller: textController, // 연결해 줍니다.
              autofocus: true,
              decoration: InputDecoration(
                hintText: "하고 싶은 일을 입력하세요",
                errorText: error,
              ),
            ),
            SizedBox(height: 32),
            // 추가하기 버튼
            SizedBox(
              width: double.infinity,
              height: 48,
              child: ElevatedButton(
                child: Text(
                  "추가하기",
                  style: TextStyle(
                    fontSize: 18,
                  ),
                ),
                onPressed: () {
                  // 추가하기 버튼 클릭시
                  String job = textController.text; // 값 가져오기
                  if (job.isEmpty) {
                    setState(() {
                      error = "내용을 입력해주세요."; // 내용이 없는 경우 에러 메시지
                    });
                  } else {
                    setState(() {
                      error = null; // 내용이 있는 경우 에러 메시지 숨기기
                    });
                    Navigator.pop(context, job); // job 변수를 반환하며 화면을 종료합니다.
                  }
                },
              ),
            ),
          ],
        ),
      ),
    ),
  );
}
}

```

## 04. 상태 관리 패키지 Provider 준비하기

### ▼ 상태 관리(State Management)의 필요성



위에서 함께 진행한 버킷 리스트 프로젝트는 페이지 수가 적고 **Bucket**에 대한 모든 CRUD(Create / Read / Update / Delete)가 **HomePage** 내에서 끝납니다.

이렇게 작은 프로젝트에서는 위에서 배우신 것과 같이 **StatefulWidget**만 활용하여도 충분히 만들 수 있습니다.



이와 달리 만약 `bucketList` 데이터를 여러 페이지에서 CRUD할 수 있는 좀 더 큰 서비스를 만들게 된다면 필연적으로 발생하는 문제가 있습니다.

예를 들어, 아래 `페이지가 많은 앱`을 보시면 Page1에는 전체 `BucketList`를 보여주고, Page2와 3에서는 조건에 따라 특정 `BucketList`만 보여주는 앱이 있다고 생각해 봅시다.

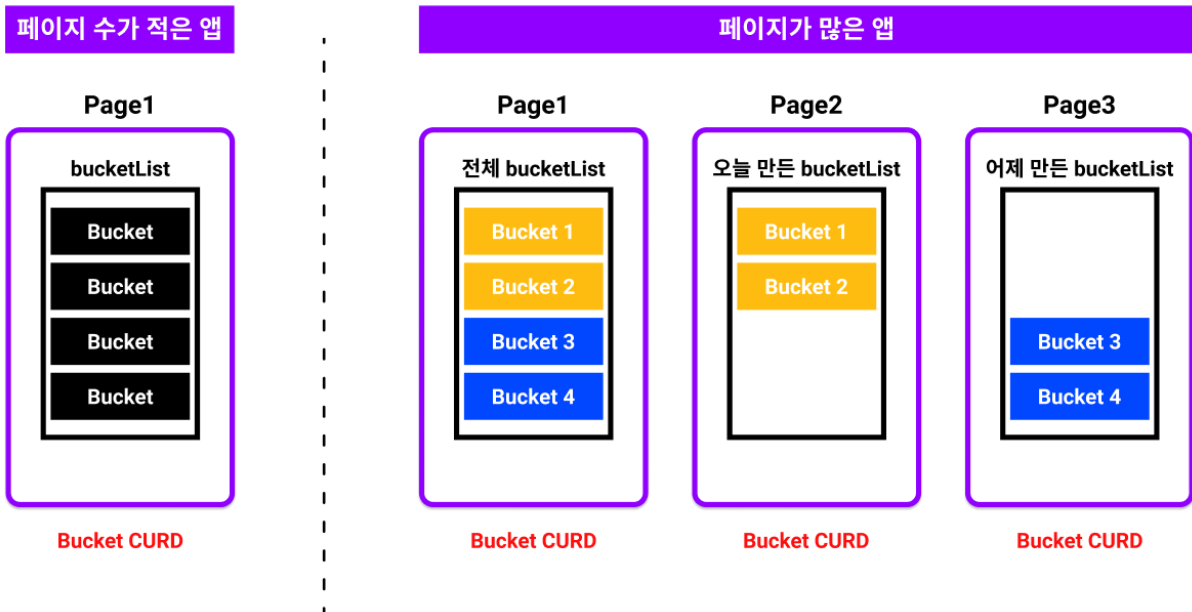
Page1에서 `Bucket1`을 삭제하면, Page2의 `Bucket1`도 삭제되어야 하고 그 반대도 마찬가지입니다. 뿐만 아니라 Page3과도 Page1도 마찬가지로입니다.

이와 같이 **최신 상태의 데이터를 보여주도록 페이지 간 데이터를 주고받고 관리하는 행위를 상태관리(State Management)**라고 부릅니다.

위와 같은 문제를 해결하기 위해 현업에서는 **상태 관리 패키지를 활용**하여 문제를 해결합니다.



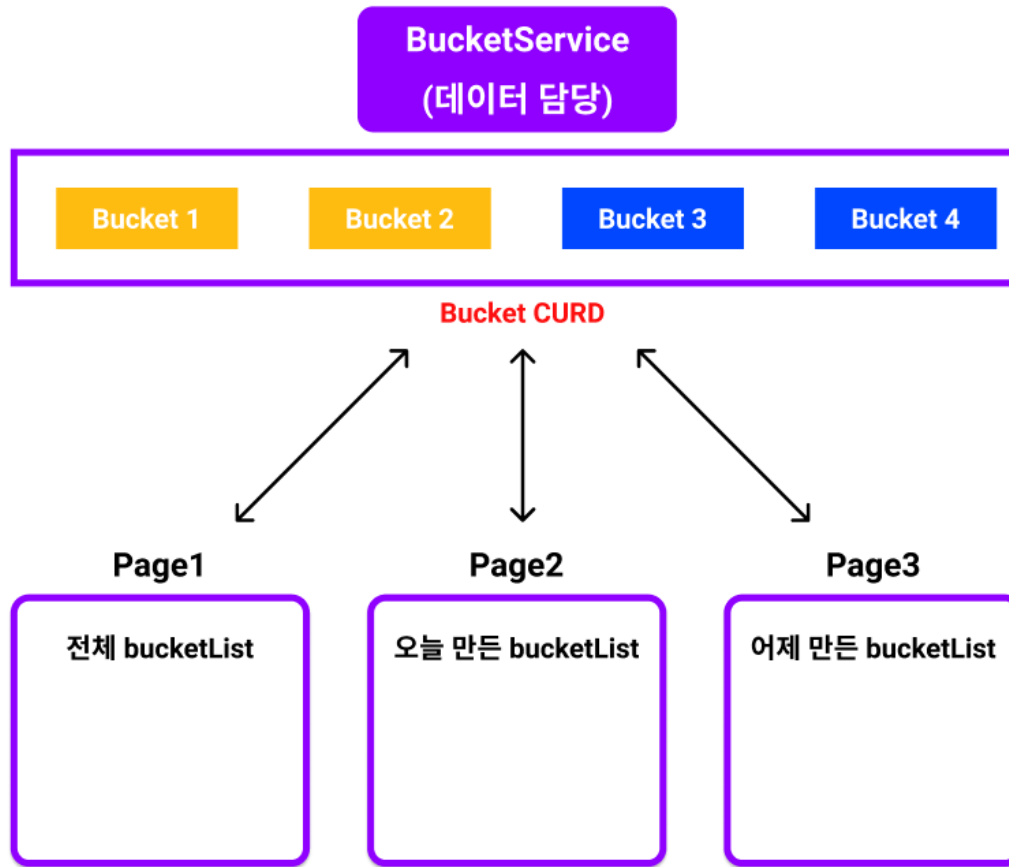
상태관리 관련 공식 문서는 [링크](#)를 참고해주세요.



위 문제의 근본적인 원인은, `BucketList`를 각 페이지에서 개별로 가지고 있기 때문입니다.

따라서 대부분의 **상태 관리 패키지**들은 아래 사진과 같이 **중앙 집중식으로 데이터를 한 곳에 모아서 관리**합니다. 앞으로 데이터를 담당하는 클래스를 **서비스(Service)**라고 부르도록 하겠습니다.

이렇게 되면 각 페이지에서는 데이터에 대한 CRUD는 모두 서비스에게 요청하는 방식으로 구현되고, 이를 통해 각 화면 간 데이터를 주고받는 문제를 해결할 수 있습니다.



💡 여러가지 상태 관리 패키지 중 **Provider**라는 패키지를 이용해서 버킷 리스트 앱을 구현해 보도록 하겠습니다.

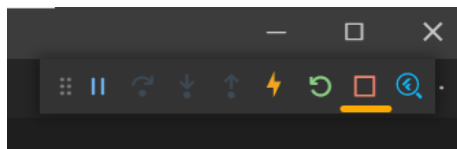
why **Provider**?

사용법이 쉽고, Flutter 공식 문서에서도 추천하는 패키지입니다.

#### ▼ 1) 프로젝트 준비

##### ▼ Flutter 프로젝트 생성

1. 기존에 실행 중인 프로젝트가 있다면 우측 상단에 빨간 네모를 눌러 종료해 주세요.



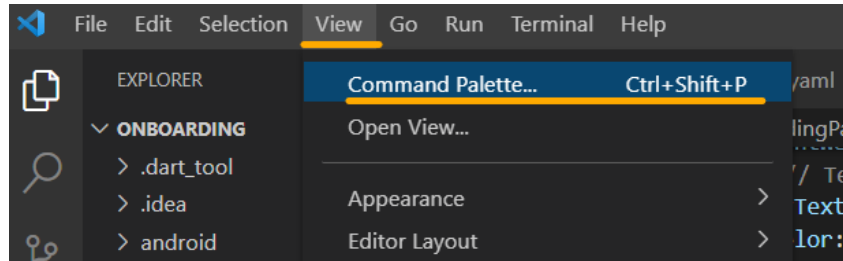
2. VSCode **View** → **Command Palette**를 선택해주세요.



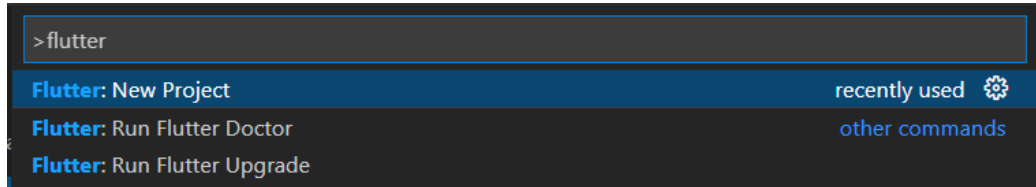
**Command Palette** 단축키

window : **Ctrl + Shift + P**

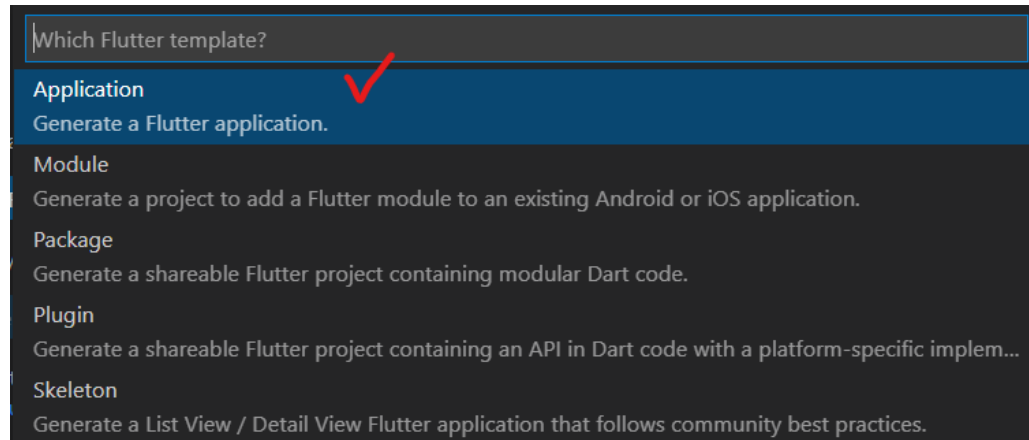
macOS : **Cmd + Shift + P**



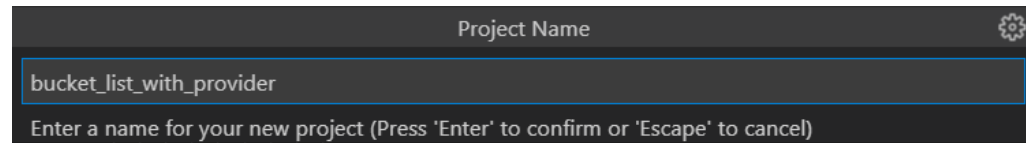
3. 명령어를 검색하는 팝업창이 뜨면, `flutter` 라고 입력한 뒤 `Flutter: New Project` 를 선택해주세요.



4. `Application` 을 선택해주세요.



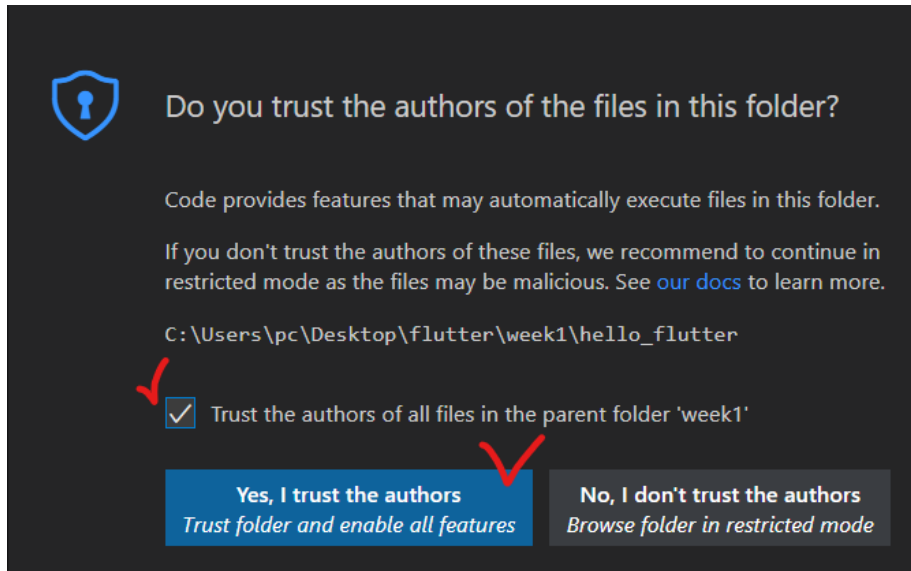
5. 프로젝트를 시작할 폴더를 선택하 화면이 나오면 `flutter` 폴더를 선택한 뒤 `Select a folder to create the project in` 버튼을 눌러 주세요.
6. 프로젝트 이름을 `bucket_list_with_provider` 로 입력해주세요.



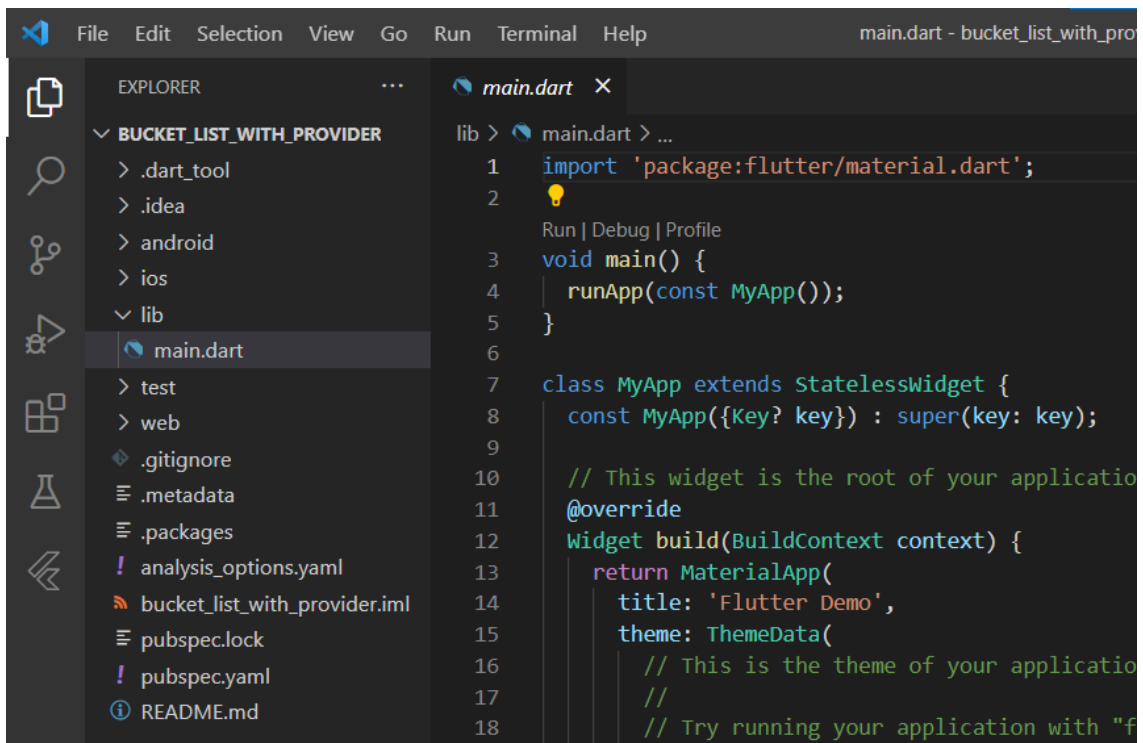
만약 중간에 아래와 같은 팝업이 뜬다면, 체크박스를 선택한 뒤 파란 버튼을 클릭해주세요. (팝업이 안보이시면 넘어가주세요!)



아래 팝업에 대한 자세한 사항은 [링크](#)를 참고해주세요.



7. 다음과 같이 프로젝트가 생성됩니다.



8. 불필요한 힌트 숨기기

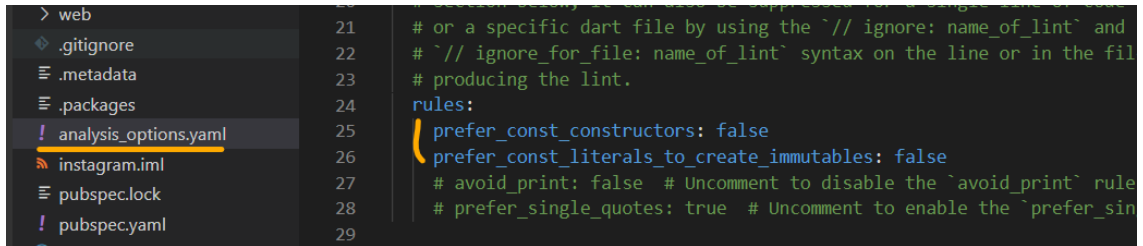
코드스니펫을 복사해서 `analysis_options.yaml` 파일에 24번째 라인 뒤에 붙여 넣어주세요.



아래 내용은 학습 단계에서 불필요한 내용을 화면에 표시하지 않도록 설정하는 과정입니다.

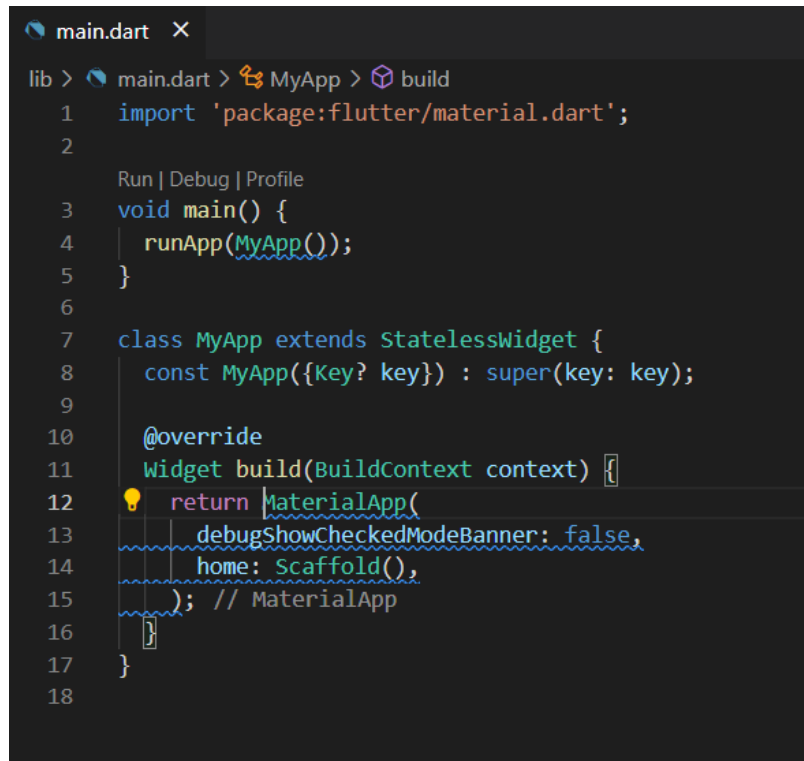
▼ [코드스니펫] `analysis_options.yaml`

```
prefer_const_constructors: false
prefer_const_literals_to_create_immutables: false
```



▼ 어떤 의미인지 궁금하신 분들을 위해

- `main.dart` 파일을 열어보시면 파란 실선이 있습니다.



- 파란 줄은, 개선할 여지가 있는 부분을 VSCode가 알려주는 표시입니다.  
12번째 라인에 마우스를 올리면 아래와 같이 설명이 뜹니다.

위젯이 변경될 일이 없기 때문에 `const` 라는 키워드를 앞에 붙여 상수로 선언하라는 힌트입니다.



상수로 만들면 어떤 이점이 있나요?

상수로 선언된 위젯들은 화면을 새로 고침 할 때 해당 위젯들은 변경을 하지 않기 때문에 스킵하여 성능상 이점이 있습니다.

아래와 같이 `Icon` 앞에 `const` 키워드를 붙여주시면 됩니다.

```

6
7 class MyApp extends StatelessWidget {
8   const MyApp({Key? key}) : super(key: key);
9
10  @override
11  Widget build(BuildContext context) {
12    return const MaterialApp(
13      debugShowCheckedModeBanner: false,
14      home: Scaffold(),
15    ); // MaterialApp
16  }
17 }
18

```

- 지금은 학습 단계이니 눈에 띄지 않도록 해주도록 하겠습니다.

9. 아래 `lib/main.dart` 코드스니펫을 복사해서 기존 코드를 모두 지우고 `main.dart` 파일에 붙여 넣은 뒤 저장해주세요.

#### ▼ [코드스니펫] main.dart

```

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

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

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      home: HomePage(),
    );
  }
}

```

```

}

/// 버킷 클래스
class Bucket {
  String job; // 할 일
  bool isDone; // 완료 여부

  Bucket(this.job, this.isDone); // 생성자
}

/// 홈 페이지
class HomePage extends StatelessWidget {
  const HomePage({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("버킷 리스트"),
      ),
      body: Center(child: Text("버킷 리스트를 작성해 주세요.")),
      floatingActionButton: FloatingActionButton(
        child: Icon(Icons.add),
        onPressed: () {
          // + 버튼 클릭시 버킷 생성 페이지로 이동
          Navigator.push(
            context,
            MaterialPageRoute(builder: (_) => CreatePage()),
          );
        },
      ),
    );
  }
}

/// 버킷 생성 페이지
class CreatePage extends StatefulWidget {
  const CreatePage({Key? key}) : super(key: key);

  @override
  State<CreatePage> createState() => _CreatePageState();
}

class _CreatePageState extends State<CreatePage> {
  // TextField의 값을 가져올 때 사용합니다.
  TextEditingController textController = TextEditingController();

  // 경고 메시지
  String? error;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("버킷리스트 작성"),
        // 뒤로가기 버튼
        leading: IconButton(
          icon: Icon(CupertinoIcons.chevron_back),
          onPressed: () {
            Navigator.pop(context);
          },
        ),
      ),
      body: Padding(
        padding: const EdgeInsets.all(16),
        child: Column(
          children: [
            // 텍스트 입력창
            TextField(
              controller: textController,
              autofocus: true,
              decoration: InputDecoration(
                hintText: "하고 싶은 일을 입력하세요",
                errorText: error,
              ),
            ),
            SizedBox(height: 32),
            // 추가하기 버튼
            SizedBox(

```



```

width: double.infinity,
height: 48,
child: ElevatedButton(
  child: Text(
    "추가하기",
    style: TextStyle(
      fontSize: 18,
    ),
  ),
),
onPressed: () {
  // 추가하기 버튼 클릭시
  String job = textController.text;
  if (job.isEmpty) {
    setState(() {
      error = "내용을 입력해주세요."; // 내용이 없는 경우 에러 메시지
    });
  } else {
    setState(() {
      error = null; // 내용이 있는 경우 에러 메시지 숨기기
    });
    Navigator.pop(context); // 화면을 종료합니다.
  }
},
),
),
),
),
);
}
}

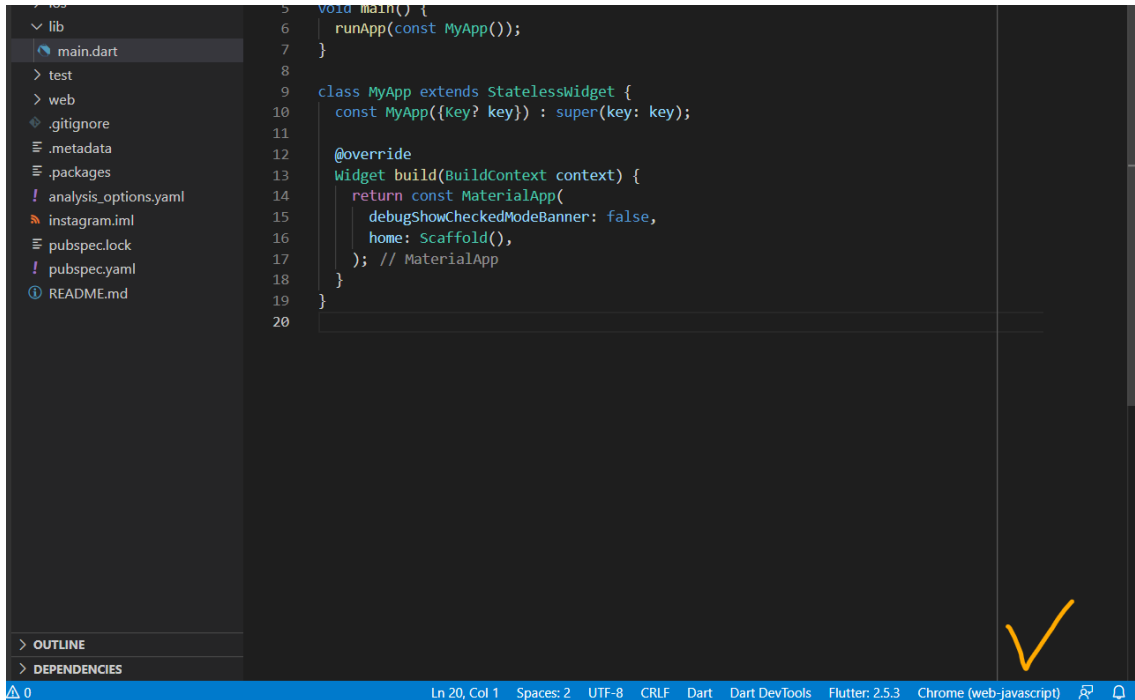
```



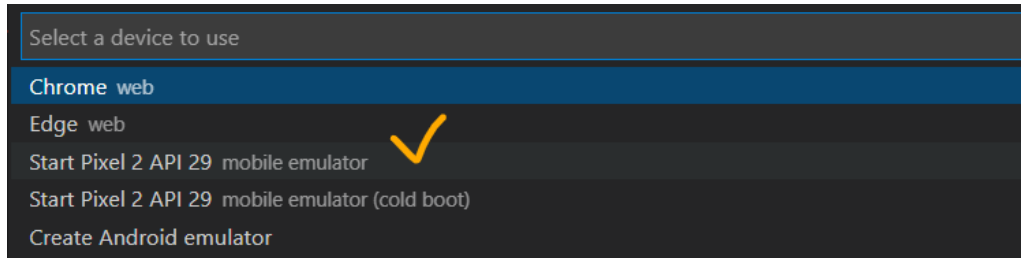
BucketList 프로젝트에서 `bucketList` 에 CRUD 하는 로직만 제외하였습니다.

#### ▼ 에뮬레이터 실행하기

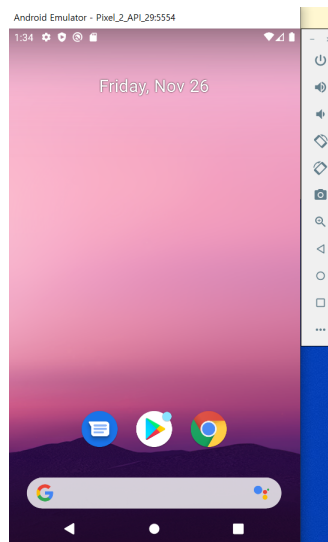
1. VSCode 우측 하단에 `Chrome (web-javascript)` 를 클릭해주세요.  
(에뮬레이터가 이미 실행중이라면 3번으로 이동해 주세요.)



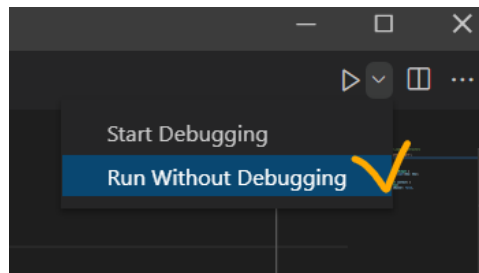
2. `Start Pixel 2 API 29 mobile emulator`를 선택해주세요. macOS의 경우 iOS 에뮬레이터를 사용하셔도 됩니다.



잠시 기다리면 에뮬레이터가 실행됩니다.



3. VSCode 우측 상단에 **아래 화살표**를 눌러 `Run Without Debugging`을 눌러주세요.



💡 `Start Debugging`으로 실행해도 무방합니다. 디버깅 모드는 특정 라인에서 앱 실행을 멈추고 해당 변수에 어떤 값이 들어있는지 볼 수 있지만 `Run without debugging`이 실행 속도가 더 빨라 안내를 위와 같이 드렸습니다 😊

에뮬레이터에 아래와 같이 버킷 리스트 화면이 나오면 완료!



## ▼ 2) **Provider** 패키지 추가

1. 코드스니펫을 복사해서 새 탭에서 열어주세요.

### ▼ [코드스니펫].pub.dev / provider

```
https://pub.dev/packages/provider/install
```

2. **Provider** 패키지 Install 탭이 아래와 같이 나오면, `flutter pub add provider` 명령어 옆에 아이콘을 눌러 복사해 주세요.

## provider 6.0.1

Published Sep 24, 2021 • [dash-overflow.net](#) Null safety

[FLUTTER](#) [ANDROID](#) [IOS](#) [LINUX](#) [MACOS](#) [WEB](#) [WINDOWS](#)

5.52K

[Readme](#) [Changelog](#) [Example](#) [Installing](#) [Versions](#) [Scores](#)

### Use this package as a library

Depend on it

Run this command:

With Flutter:

```
$ flutter pub add provider
```

This will add a line like this to your package's pubspec.yaml (and run an implicit `flutter pub get`):

3. `View` → `Terminal` 을 열어 주세요.