3학년 1학기 앱프로그래밍응용 중간고사 정리본

# Flutter 시험범위별 교재 페이지 정리

시험: 30문항 사지선다형 (1문항 당 1점, 총 30점)

시험 시간: 1시간

## 0장: 플러터 개발 환경 준비하기 (2문항)

|  |  |  |  |
| --- | --- | --- | --- |
| 주제 | 소제목 | 페이지 | 비고 |
| 앱 개발 방식 | 0.1 앱 개발 방식 소개 | p.24 |  |
| 플러터 특징 | 0.2 플러터 소개 | p.25 |  |

## 1장: 생애 첫 모바일 앱 생성하기 (1문항)

|  |  |  |  |
| --- | --- | --- | --- |
| 주제 | 소제목 | 페이지 | 비고 |
| 환경 구성 검사 | 0.3 개발 환경 구성 | p.26 | → 실제로는 1장 범위지만 책상 0장에 포함됨 |

## 2장: 다트 문법 (10문항)

|  |  |  |  |
| --- | --- | --- | --- |
| 주제 | 소제목 | 페이지 | 비고 |
| 변수타입 | 2.2 기본 문법 | p.62 |  |
| final, const | 2.3 변수와 상수 | p.64 |  |
| 타입검사 / 형변환 | 2.4 분기문, 반복 | p.67 | 강의 중 설명한 것으로 출제됨  힌트: 상속, LLM활용 |
| 익명함수, 람다식 | 2.7 함수형 프로그래밍 | p.72 |  |
| 선택 매개변수 | 2.7 함수형 프로그래밍 | p.73 |  |
| 클래스 정의 | 2.5 객체 지향 프로그래밍 | p.76 |  |
| 상속 vs 인터페이스 | 2.5 객체 지향 프로그래밍 | p.80 ~ | 강의 설명 기반 출제  힌트: 서로 장단점이 반대임, LLM 활용 |
| List, Map, Set 표기법 | 2.6 컬렉션 | p.84 |  |
| 함수형 프로그래밍 특징 | 2.7 함수형 프로그래밍 | p.87 | 일급 객체, forEach, where, map |
| 컬렉션 if/for | 2.8 기타 유용한 기법 | p.95 |  |

## 3장: 프로젝트 구조와 앱 구조 (2문항)

|  |  |  |  |
| --- | --- | --- | --- |
| 주제 | 소제목 | 페이지 | 비고 |
| pubspec.yaml 설명 | 3.1 프로젝트 구조 이해 | p.100 | LLM 활용하여 상세 이해 필요 |
| 샘플 앱 기본구조에서 출제 | 3.2 샘플 앱 분석하기 | p.101 | EX)Stateless/Stateful 기본 구조 및 동작 차이 |

## 4장: 기본 위젯 I (6문항)

|  |  |  |  |
| --- | --- | --- | --- |
| 주제 | 소제목 | 페이지 | 비고 |
| 화면 배치 위젯 | 4.2 화면 배치에 쓰는 기본 위젯 | p.126 | 2문항 |
| 위치/정렬/크기 위젯 | 4.3 위치, 정렬, 크기 위젯 | p.141 | 2문항 |
| 버튼 등 위젯 | 4.4 버튼 계열 위젯 | p.149 | 2문항 |

## 5장: 기본 위젯 II (5문항)

|  |  |  |  |
| --- | --- | --- | --- |
| 주제 | 소제목 | 페이지 | 비고 |
| 입력 | 5.1 입력용 위젯 | p.160 |  |
| 다이얼로그 | 5.2 다이얼로그 | p.168 |  |
| 이벤트 | 5.3 이벤트 | p.174 |  |
| 애니메이션 | 5.4 애니메이션 | p.177 |  |
| 쿠퍼티노 디자인 | 5.5 쿠퍼티노 디자인 | p.185 |  |

## 6장: 내비게이션 (3문항)

|  |  |  |  |
| --- | --- | --- | --- |
| 주제 | 소제목 | 페이지 | 비고 |
| push/pop 사용법 | 6.1 새로운 화면으로 이동 | p.194 |  |
| routes 사용법 | 6.2 routes를 활용한 내비게이션 | p.201 | 강의 설명 기반 출제, LLM 학습 권장 |
| 내비게이션 동작 방식 | 6.3 내비게이션 동작 방식 이해 | p.202 |  |

## 8장: Flutter 2 변경점 (1문항)

|  |  |  |  |
| --- | --- | --- | --- |
| 주제 | 소제목 | 페이지 | 비고 |
| Null Safety | 8.1 널 안전성 | p.242 | Flutter 3 기준 Null-safe 코드 숙지 필요 (LLM 활용) |

📘 0장

✅ 0.1 앱 개발 방식 소개 (p.24) [출제: 1문항]

- Native 방식: 플랫폼별로 따로 개발 (Android: Java/Kotlin, iOS: Swift/Objective-C)

- Hybrid 방식: 웹 기술 기반 앱 (예: Cordova), 성능과 UI 일관성 부족

- Cross-platform 방식: 하나의 코드로 여러 플랫폼 대응 (예: Flutter)

→ Flutter는 Cross-platform 방식에 해당

✅ 0.2 플러터 소개 (p.25) [출제: 1문항]

- 한 번의 코드로 여러 플랫폼(Android, iOS, Web, Desktop 등) 지원

- 장점:

• 낮은 진입장벽

• 높은 성능 (UI를 직접 그림 → 네이티브에 가까움)

• 예쁜 UI 제공

• Hot Reload 등 생산성 높은 개발 도구 지원

- 디자인 시스템 지원:

- 안드로이드: 머터리얼 디자인

- iOS: 쿠퍼티노 디자인

- Dart 언어 사용 (자바/자바스크립트와 유사)

📘 1장

✅ 0.3 개발 환경 구성 (p.26~31) [출제: 1문항 → 1장 환경 구성 검사로 출제]

- 설치 순서

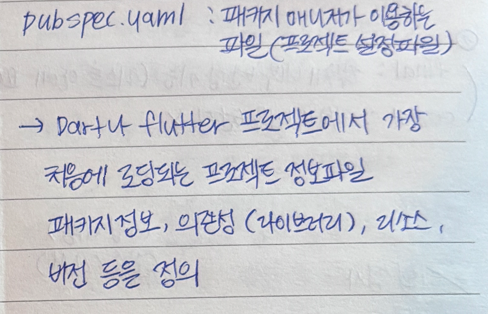
- Flutter SDK 설치

- 환경 변수 등록

- 개발 도구 설치

Flutter doctor 명령 – 플러터를 개발할 환경 구성 검사

1. 터미널에서 플러터 명령 실행
2. 안드로이드 스튜디오 하단 터미널 탭 누른 후 실행 -> 직접 flutter doctor 입력
3. 왼쪽 프로젝트 창에서 flutter-app > test > pubspec.yaml 선택하고, 오른쪽 상단에 표시된 4가지 링크 중 flutter doctor 클릭



📘 2장

✅ 2.2 기본 문법 – 변수 타입 (p.62)

- 다트에서 기본이 되는 데이터 타입:  
 • int: 정수형  
 • double: 실수형  
 • String: 문자열  
 • bool: true 또는 false를 나타내는 논리형  
- 숫자형 타입을 포괄하는 타입으로 num이 있으며, int와 double을 포함한다.  
- var를 사용하면 타입을 생략하고 자동 추론 가능하다.

예시 코드:

int a = 10;  
double b = 2.5;  
num c = a + b;  
var name = '홍길동';

✅2.3 변수와 상수 – final, const (p.64)

- final: 값을 한 번만 할당할 수 있으며, 런타임 시 결정됨/프로그램을 실행할 때 값이 설정  
- const: 컴파일 시점에 상수로 결정되어 완전히 고정됨./프로그램 실행 전 정해짐(완전불변)  
- const > final > var 순으로 제약이 강함.

예시 코드:

final String name = '홍길동';  
const int age = 20;

✅ 2.4 분기문, 반복 – 타입검사 / 형변환 (p.67) -> 강의 중 설명 출제

- 객체의 타입을 확인하거나 변환할 때 사용하는 키워드:  
 • is: 객체가 특정 타입인지 확인 (true/false 반환)  
 • is!: 특정 타입이 아님을 확인

예시 코드

var v1 = 5; // 정수이므로 int, num의 하위 타입

var v2 = 3.1415; // 실수이므로 double, num의 하위 타입

var v3 = "문자열";

var v4 = '문자열2';

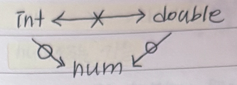
var v5 = true;

print(v1 is num); // true

print(v2 is num); // true

print(v1 is int); // true

print(v2 is double); // true

• as(형변환): 타입 변환에 사용  
- 상속 관계에서 자식 객체를 부모 타 입으로 변환하거나 검사할 때 자주 사용됨. (상위 개념만 가능)   


Dart는 강한 타입 언어로, 명확한 타입 체크와 변환이 중요함. (다른 타입끼리 변환 x)

예시 코드:

int a = 10;

num b = a as num; // a를 num 타입으로 **형변환 int는 num의 하위 타입이기 때문에 문제 없이 변환**

print(b) // 10출력

✅ 2.7 함수형 프로그래밍 – 익명함수, 람다식 (p.72)

- 이름 없이 정의되는 함수로, 일회성 동작에 자주 사용됨.  
- 기본 구조: (매개변수) { 실행코드; }  
- 람다식은 표현식만 작성할 수 있으며 => 기호를 사용함.

- 기본 구조: (매개변수) => 실행코드; 주로 콜백 함수나 반복문 안에서 활용됨.

✅ 2.7 함수형 프로그래밍 – 선택 매개변수 (p.73)

- 함수 정의 시 {}로 감싼 매개변수를 선택적으로 사용.  
- 호출 시 매개변수명을 값 앞에 사용 -> 이름있는매개변수

- 필수매개변수, 선택매개변수 샅이 사용 시 필수 매개변수를 앞쪽에 작성

- 기본값도 함께 지정 가능하여 함수 호출 시 유연하게 처리할 수 있음.

예시 코드:

void greet(String name, {String greeting = '안녕하세요'}) {  
 print('$greeting, $name');  
}  
  
greet('홍길동');  
greet('이몽룡', greeting: '반갑습니다');

✅ 2.5 객체 지향 프로그래밍 – 클래스 정의 (p.76)

|  |  |
| --- | --- |
|  | - 클래스는 일종의 사용자 정의 타입으로 설계도 역할로 변수(속성)와 함수(기능)를 묶어 정의  => 프로퍼티: 클래스 내부의 속성 값 - 인스턴스는 클래스에서 실제로 생성된 객체**.** - 객체는 저장 공간에 할당 된 값, 식별자에 의해 참고되는 공간  =>변수, 자료 구조, 함수, 메서드 등 |

예시 코드:

class Person { // 클래스 선언  
 String name; // 프로퍼티: 이름  
 int age; // 프로퍼티: 나이  
  
 Person(this.name, this.age); // 생성자  
  
 void introduce() { // 메서드  
 print('저는 $name, $age살입니다.');  
 }  
}  
  
var p = Person('홍길동', 20); // 객체 생성 및 메서드 호출  
p.introduce(); // 출력: 저는 홍길동, 20살입니다.

✅ 2.5 객체 지향 프로그래밍 – 상속 vs 인터페이스 (p.80~)

| **항목** | **extends (상속)** | **implements (인터페이스 구현)** |
| --- | --- | --- |
| **용도** | 부모 클래스의 기능을 물려받아 재사용 | 인터페이스를 형식만 가져와 직접 구현 |
| **구문** | class SubClass extends SuperClass | class MyClass implements Interface1, Interface2 |
| **재정의 여부** | 선택적으로 재정의 가능 (@override 사용) | 모든 메서드를 **반드시 재정의**해야 함 |
| **다중 사용** | ❌ 단일 상속만 가능 | ✅ 여러 인터페이스 구현 가능 |
| **주 목적** | 코드 **재사용 / 기능 확장** | 계약(규격) 정의 / **명세 기반 개발** |
| **상속 대상** | 구체 클래스 또는 추상 클래스 | 추상 클래스 또는 일반 클래스 모두 가능 |
| **✅ 장점** | - 기존 기능 재사용 가능 - 유지보수 효율적 | - 명확한 계약 기반 개발 - 다중 구현 가능 - 유연한 구조 설계 가능 |
| **❌ 단점** | - 다중 상속 불가 - 유연성이 떨어질 수 있음 | - 기능 직접 구현 필요 - 코드 중복 가능성 있음 |

✅ 2.6 컬렉션 – List, Map, Set 표기법 (p.84~)

- List: 순서가 있는 컬렉션, 인덱스를 이용해 접근 가능  
- Set: 중복을 허용하지 않는 컬렉션  
- Map: 키와 값의 쌍으로 이루어진 구조

예시 코드:

List<String> fruits = ['사과', '바나나', '딸기'];  
Set<int> numbers = {1, 2, 3};  
Map<String, int> scores = {'철수': 90, '영희': 85};

✅ 2.7 함수형 프로그래밍 – 함수형 특징 (p.87~91)

• forEach: 각 요소에 대해 내부 반복 수행  
• where: 조건에 맞는 요소 필터링

• map: 반복되는 값을 다른 형태로 변환하는 방법을 제공하는 함수

• toList, toSet: 리스트/셋으로 변환 가능

• 일급객체: 함수를 값으로 취급 -> 다른 변수에 함수 대입 가능

예시 코드:

List<int> numbers = [1, 2, 3, 4, 5];  
numbers.forEach((n) => print(n)); // 1. forEach: 각 요소 출력  
var even = numbers.where((n) => n % 2 == 0); // 2. where: 짝수만 필터링  
var doubled = numbers.map((n) => n \* 2).toList(); // 3. map + toList: 각 요소를 2배로 바꾸고 리스트로 저장

✅ 2.8 기타 유용한 기법 – 컬렉션 if/for (p.95~96)

| **구분** | **설명** | **예시** |
| --- | --- | --- |
| **컬렉션 if** | 리스트나 Set 안에서 **조건부로 항목을 포함**할 수 있음 | if (조건) 값 |
| **컬렉션 for** | **기존 컬렉션을 반복하면서** 새 컬렉션에 요소 추가 | for (var x in 리스트) 처리식 |

# isLoggedIn = false일 경우: [Home, Profile]만 출력됨

void main() {

bool isLoggedIn = true;

var menu = [

'Home',

'Profile',

if (isLoggedIn) 'Logout', // 조건이 true일 때만 추가

];

print(menu); // 출력: [Home, Profile, Logout]

}

# numbers 리스트를 반복하면서 각 요소에 2를 곱해 새 리스트를 만듬.

void main() {

var numbers = [1, 2, 3];

var doubleNumbers = [

for (var n in numbers) n \* 2

];

print(doubleNumbers); // 출력: [2, 4, 6]

}

📘 3장

✅ 3.1 pubspec.yaml 관련 설명 (!주의! LLM을 이용해 상세히 파악할 것!)

- pubspec.yaml은 Flutter 프로젝트의 "설정서"로, 앱이 작동하는 데 필요한 모든 구성 요소들을 정의해주는 핵심 파일

- 패키지, 이미지, 폰트 등 앱에서 사용하는 리소스와 설정을 관리하는 파일

**📌 주의사항**

* 들여쓰기는 반드시 **스페이스 2칸**을 사용하세요. (탭 ❌)
* 변경 후에는 항상 flutter pub get 실행이 필요합니다.
* 경로 지정은 프로젝트 폴더 기준입니다 (assets/, fonts/ 등).

✅ pubspec.yaml 구성요소 상세 정리표

| **항목** | **설명** | **작성 예시** | **비고 / 용도** |
| --- | --- | --- | --- |
| name | 프로젝트 이름 | name: my\_app | 패키지나 앱의 식별자 역할 |
| description | 앱/패키지 설명 | description: A Flutter app | pub.dev에 업로드 시 표시됨 |
| version | 버전 + 빌드 번호 | version: 1.0.0+1 | 앱스토어 배포 시 중요 |
| environment | SDK 버전 조건 | sdk: ">=3.0.0 <4.0.0" | 사용 가능한 Dart SDK 범위 지정 |
| dependencies | 실행 시 필요한 외부 패키지 | http: ^0.14.0 flutter: sdk: flutter | flutter pub get 명령으로 설치 |
| dev\_dependencies | 개발 시에만 필요한 패키지 | flutter\_test: sdk: flutter | 테스트, 빌드 도구 등 (앱 실행에는 불필요) |
| flutter: | Flutter 전용 설정 블록 | uses-material-design: true | 머티리얼 아이콘, 에셋, 폰트 등 포함 |
| assets | 리소스(이미지, json 등) 경로 | - assets/images/ - assets/data/info.json | flutter: 아래에 위치. 폴더 끝은 / 필요 |
| fonts | 커스텀 폰트 설정 | yaml<br>- family: Nanum<br> fonts:<br> - asset: assets/fonts/Nanum-Regular.ttf<br> | TextStyle(fontFamily: 'Nanum') 으로 사용 |
| uses-material-design | 머티리얼 아이콘 사용 여부 | uses-material-design: true | 사용 시 Icons.xxx 사용 가능 |
| module | 모듈 설정 (플러터 모듈일 경우) | androidPackage: com.example.myapp | 기존 앱에 Flutter 삽입 시 필요 |
| plugin | 플러그인 프로젝트 설정 | platforms: 하위 설정 | 플러터 플러그인 개발 시 사용 |

✅ 3.2 샘플 앱 분석하기 ex) Stateless/Stateful 기본 구조 및 동작 차이 (p.101)

- main() 함수에서 runApp()으로 앱이 시작됨.

- StatelessWidget은 상태 변화 없이 **한 번 그리고 끝나는 UI**,  
- StatefulWidget은 **사용자 입력 등으로 UI가 바뀌는 동적인 화면**을 만들 때 사용

| **항목** | **StatelessWidget** | **StatefulWidget** |
| --- | --- | --- |
| **상태 변경** | ❌ 상태 변화 없음 | ✅ 상태 변화 있음 |
| **사용 용도** | 고정된 UI (예: 로고, 타이틀) | 사용자 입력, 버튼 클릭 등 반응형 UI |
| **구조** | build() 메서드만 존재 | createState() → State 클래스 내부에 build() |
| **UI 갱신 방법** | 변경 불가 (다시 빌드하려면 새 인스턴스 생성) | setState() 호출 시 UI 자동 갱신 |
| **성능** | 가벼움 | 무거움 (상태 저장 필요) |

MyApp은 StatelessWidget  
→ 앱의 기본 구성 (MaterialApp, 첫 페이지 등)을 정의

MyHomePage는 StatefulWidget  
→ 버튼을 누르면 setState() 호출로 \_counter 값이 증가하고 build() 재실행

예시 코드 :

```dart

import 'package:flutter/material.dart';

// 앱의 시작점

void main() => runApp(MyApp());

// 상태가 없는 위젯: 앱 전체 구조 정의

class MyApp extends StatelessWidget {

@override

Widget build(BuildContext context) {

return MaterialApp(

home: MyHomePage(),

);

}

}

// 상태가 있는 위젯: 사용자 인터랙션에 따라 UI 갱신 가능

class MyHomePage extends StatefulWidget {

@override

\_MyHomePageState createState() => \_MyHomePageState();

}

class \_MyHomePageState extends State<MyHomePage> {

int \_counter = 0; // 클릭 횟수를 저장할 변수

void \_incrementCounter() {

setState(() {

\_counter++; // 상태 변경 발생 → UI 갱신됨

});

}

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(title: Text('홈')),

body: Center(

child: Column(

mainAxisAlignment: MainAxisAlignment.center,

children: <Widget>[

Text('버튼 클릭 횟수:'),

Text('$\_counter'),

],

),

),

floatingActionButton: FloatingActionButton(

onPressed: \_incrementCounter, // 버튼 클릭 시 동작

child: Icon(Icons.add),

),

);

}

}

📘 4장

✅ 1. 화면 배치 위젯 126p

4.2.1 Container

- 가장 기본적인 레이아웃 위젯

- 가르기, 색상, 어겨운 외부 영역(padding/margin)을 포함한 다양한 프로퍼티 제공

- child 위젯 포함 가능

Container(

color: Colors.red,

width: 100,

height: 100,

padding: EdgeInsets.all(8.0),

margin: EdgeInsets.all(8.0),

)

4.2.2 Column / 4.2.3 Row

- Column: 수직 방향, Row: 수평 방향 포건

- children 리스트에 여러 위젯 배치

- mainAxisAlignment, crossAxisAlignment 여러 정렬 가능

Row(

mainAxisAlignment: MainAxisAlignment.center,

crossAxisAlignment: CrossAxisAlignment.center,

children: [ ... ],

)

4.2.4 Stack

위젯을 겹칠 수 있는 위젯 (z축 순서)

예시: 빨간, 초록, 파란색 박스 겹칭

Stack(

children: [Container(...), Container(...), ...],

)

4.2.5 SingleChildScrollView

- 화면이 넘치면 스크롤 가능

- Column, ListBody 같은 배치와 같이 사용

SingleChildScrollView(

child: Column(children: [...]),

)

4.2.6 ListView / ListTile

- ListView: 여러 위젯을 리스트형으로 표시

- ListTile: leading/icon, title(제목), trailing(복수 정보), onTap

ListView(

children: [

ListTile(

leading: Icon(Icons.home),

title: Text('Home'),

trailing: Icon(Icons.navigate\_next),

onTap: () {},

),

],

)

4.2.7 GridView

- 열 수를 지정해 그리드 형포 표시

GridView.count(

crossAxisCount: 2,

children: [...],

)

4.2.8 PageView

- 여러 페이지를 슬라이드로 넘겨가는 위젯

- Tab과 같이 사용할 경우 더 많이 사용

PageView(

children: [...],

)

4.2.9 AppBar, TabBar, TabBarView

- AppBar 하단에 tabs 설정

- DefaultTabController가 필요

- TabBarView와 연동

DefaultTabController(

length: 3,

child: Scaffold(...)

)

4.2.10 BottomNavigationBar

- 하단에 2~5개 탭 구성

- BottomNavigationBarItem(icon + label) 및 onTap 이벤트

✅ 2. 위치/정렬/크기 위젯 141p

📍 4.3.1 Center

* child 위젯을 화면 중앙에 배치

📍 4.3.2 Padding

* 안쪽 여백

📍 4.3.3 Align

* 자식 위젯의 정렬 방향을 정함

📍 4.3.4 Expanded

* 자식 위젯의 크기를 최대한 확장
* flex 값 설정 (default: 1)

Expanded(

flex: 2,

child: Container(...),

)

📍 4.3.5 SizedBox

* 크기 프로퍼티가 없는 위젯의 크기 조정
* width, height 값 지정

📍 4.3.6 Card

* 카드 형태 모양을 제공하는 위젯

✅ 3. 버튼 등 위젯 149p

🔘 4.4.1 ElevatedButton

입체감을 가지는 일반적인 버튼, onPressed 필수

ElevatedButton(

child: Text('RaisedButton'),

onPressed: () {},

)

🔘 4.4.2 TextButton

평평한 형태의 버튼

TextButton(

child: Text('TextButton'),

onPressed: () {},

)

🔘 4.4.3 IconButton

아이콘을 표시하는 버튼

icon, iconSize, onPressed

🔘 4.4.4 FloatingActionButton

입체감 있는 둥근 버튼

✅ 4. 화면 표시용 위젯

🔘 4.5.1 Text

텍스트 표시용 위젯

TextStyle: fontSize, fontWeight, fontStyle, color, letterSpacing

Text(

'Hello World',

style: TextStyle(fontSize: 40.0, fontWeight: FontWeight.bold),

)

🔘 4.5.2 Image

Image.network() / Image.asset()

assets 폴더, pubspec.yaml 설정 필요

🔘 4.5.3 Icon

아이콘을 다루는 단일 위젯 (color, size 지정 가능)

🔘 4.5.4 Progress

로딩을 보여주는 위젯

CircularProgressIndicator / LinearProgressIndicator

🔘 4.5.5 CircleAvatar

웹사진, 프로필 등에 사용하는 원형위젯

child과 backgroundImage 설정 가능

📘 5장

✅ 5.1 입력용 위젯 (1문항) 160p

🔘 TextField: 글자를 입력받는 위젯. InputDecoration으로 외곽선 및 힌트 메시지 설정 가능.  
🔘 CheckBox & Switch: boolean 값에 따라 UI 상태 변경. StatefulWidget 필수.

🔘 Radio & RadioListTile: 선택 그룹중 하나를 선택할 때 사용

🔘 DropDownButton: 여러 아이템 중 하나를 고를 수 있는 콤보박스 형태의 위젯

✅ 5.2 다이얼로그 (1문항) 168p

용도 : 사용자의 확인 요구 / 메시지 표시

- AlertDialog: 유저확인용 다이얼로그. 제목(title), 내용(content), 버튼(actions)으로 구성.  
- DatePicker: 날짜 선택용 다이얼로그.

- TimePicker: 시간선택

✅ 5.3 이벤트 (1문항) 174p

: onTap, onPressed 등의 이벤트를 기본 프로퍼티로 가지고 있지 않은 위젯에 이벤트를 적용할 수 있도록 해주는 위젯  
- GestureDetector & InkWell: 글자나 그림 같이 이벤트 프로퍼리가 없을 때

* 터치이벤트 발생.
* 클릭 이벤트 처리 가능.
* InkWell은 물결 효과 제공.

✅ 5.4 애니메이션 (1문항) 177p

- Hero: 화면 전환 시 이미지 등 자연스러운 전환 가능.  
- AnimatedContainer: setState와 함께 화면을 새로 그릴 때 애니메이션 적용.  
- SliverAppBar, SliverFillRemaining: 화면 헤더를 동적으로 표현

- SliverAppBar, SliverList: 리스트 표현

✅ 5.5 쿠퍼티노 디자인 (1문항) 177p

- iOS 스타일의 위젯: CupertinoNavigationBar, CupertinoSwitch, CupertinoAlertDialog 등.

- CupertinoPicker: showCupertinoModalPopup과 함께 사용

📘 6장

✅ push / pop 사용법 (196p, 198p)

- Navigator.push(): 새로운 화면으로 이동  
- Navigator.pop(): 이전 화면으로 되돌아가기  
- MaterialPageRoute를 통해 연결되는 화면 정의  
- push는 Future 타입 반환 → await로 결과 처리 가능

```dart  
Navigator.push(  
 context,  
 MaterialPageRoute(builder: (context) => SecondPage()),  
);  
  
Navigator.pop(context); // 현재 페이지 종료  
```

✅ routes 사용법 (201p)

- routes는 MaterialApp 내에서 앱 전체 화면 경로를 전역 설정할 수 있음  
- 각 라우트는 Map 형식으로 정의하며, pushNamed()를 통해 호출함  
- 화면 전환 로직을 단순화하고 구조화할 수 있어 실제 앱에서 많이 사용됨  
- 실무에서도 URL 기반의 네비게이션과 연결되는 중요한 개념

```dart  
MaterialApp(  
 routes: {  
 '/first': (context) => FirstPage(),  
 '/second': (context) => SecondPage(),  
 },  
);  
  
// 사용  
Navigator.pushNamed(context, '/second');  
```

✅ 네비게이션 동작 방식 (202p)

- Navigator는 Stack 구조로 작동 (push로 추가, pop으로 제거)  
- StatelessWidget: build()만 실행됨  
- StatefulWidget: initState() → build() → dispose() 순으로 생명주기 동작  
- push/pop 흐름을 통해 생명주기 순서를 직접 확인 가능

```dart  
@override  
void initState() {  
 super.initState();  
 print('initState');  
}  
  
@override  
void dispose() {  
 super.dispose();  
 print('dispose');  
}  
```

📘 8장 Flutter 3 기준 Null Safety 정리

**Null Safety**는 Dart 2.12부터 도입된 기능으로, \*\*Flutter 3에서는 기본값(필수 적용)\*\*으로 사용 됨

✅ Null Safety란?

**null 오류를 방지하기 위한 시스템**

개발자가 \*\*"이 변수는 null이 될 수 있다/없다"\*\*를 **명시**하도록 강제함

**컴파일 단계에서 null 관련 오류를 체크**해서 **미리 방지**

✅ 1. Null Safety 기본 개념

**기본적으로 변수는 null을 허용하지 않음**

int a = 10; // 가능

int b = null; // 오류 발생

**null을 허용하려면 ?를 붙여야 함**

int? c = null; // 가능

✅ 2. 주요 키워드 및 연산자



📌 ? (nullable type 선언)

String? name = null;

📌 ! (non-null 단언 연산자)

"절대 null 아님"을 보장할 때 사용

String? name;

print(name!); // 런타임 오류 가능성 있음

📌 ?? (null 병합 연산자)

null이면 기본값을 지정

String? name;

print(name ?? '익명');

📌 ?. (null-aware 접근 연산자)

String? name;

print(name?.length); // name이 null이면 null 반환

✅ 3. late 키워드

**나중에 초기화할 변수를 선언할 때 사용**

null을 허용하지 않으면서도 선언 시 값을 줄 수 없을 때

late String title;

title = "Flutter"; // 나중에 초기화 가능

✅ 4. null 값 처리 예시

int? age;

print(age ?? 0); // age가 null이면 0 출력

String? text;

print(text?.toUpperCase());

✅ 5. Nullable 타입과 기본 타입 비교

| **타입** | **설명** |
| --- | --- |
| int | null 불가 |
| int? | null 허용 |
| String | null 불가 |
| String? | null 허용 |

✅ 6. Null Safety 장점

런타임 에러 감소 (컴파일 타임에 미리 확인)

안정성 높은 코드 작성 가능

코드 가독성과 유지보수성 향상

✅ 7. Null Safety 적용 조건 (Flutter 3)

Dart SDK가 2.12 이상 ("sdk": ">=2.12.0 <3.0.0")

null safety 기반 패키지 사용 필요

environment:

sdk: ">=2.12.0 <3.0.0"

✅ 8. 마이그레이션 명령어

기존 프로젝트를 null-safe하게 바꾸려면:

dart migrate