MOJADIPRO

STRUKTUR APLIKASI FLUTTER

MOJADIPRO

MATERI

01

Pelajari tentang cara kerja Lists dan Conditionals di Dart.

02

Pelajari tentang kelas dan objek di Dart.

03

Memahami Object Oriented Dart dan cara menerapkan fundamental OOP.

04

Pelajari cara menggunakan Dart Constructors.

05

Pelajari cara menggunakan Flutter Themes & The Gesture Detector Widget

MOJADIPRO

MATERI

06

Pelajari tentang maps, enums, dan ternary operator di Dart.

07

Perbedaan antara final dan const di Dart.

08

Immutability Stateless dan Stateful Widget.

09

Flutter Favours Composition vs Inheritance.

MOJADIPRO

LIST

List (daftar) adalah kumpulan nilai atau objek yang diatur dalam urutan tertentu. Dart List mirip dengan array, tetapi memiliki fleksibilitas yang lebih tinggi dan lebih mudah digunakan.

Dalam Dart, List dapat berisi objek dari jenis apa pun, seperti angka, string, boolean, atau bahkan objek yang dibuat oleh pengguna sendiri.
List dapat diinisialisasi dengan menggunakan literal List atau dengan menggunakan konstruktor List().

MOJADIPRO

LIST

```
// membuat list kosong dengan ukuran 10
var namaList = new List(10);

// membuat list kosong yang tak jelas ukuran panjangnya
var namaList = [];

// membuat list sekaligus mengisinya dengan nilai
var namaList = ["Belajar", "Dart", "itu", "mengesankan"];
```

MOJADIPRO

LIST

Cara Kerja Lists di Dart:

1. Dengan menggunakan sintaksis List<T>() atau menggunakan literal list [] atau List<T>[]

```
List<int> numbers = List<int>(); // Membuat list kosong dengan tipe data int
List<String> names = []; // Membuat list kosong dengan tipe data String menggunakan literal list
List<double> prices = List<double>(); // Membuat list kosong dengan tipe data double
```

MOJADIPRO

LIST

2. Menambahkan dan mengakses elemen menggunakan metode add()

```
List<int> numbers = [1, 2, 3];
numbers.add(4); // Menambahkan elemen 4 ke list numbers
print(numbers[0]); // Mengakses elemen pertama dari list (output: 1)
```

MOJADIPRO

LIST

3. Menghapus elemen menggunakan metode remove() atau removeAt()

```
List<int> numbers = [1, 2, 3, 4];
numbers.remove(3); // Menghapus elemen 3 dari list
numbers.removeAt(0); // Menghapus elemen pertama dari list
```

MOJADIPRO

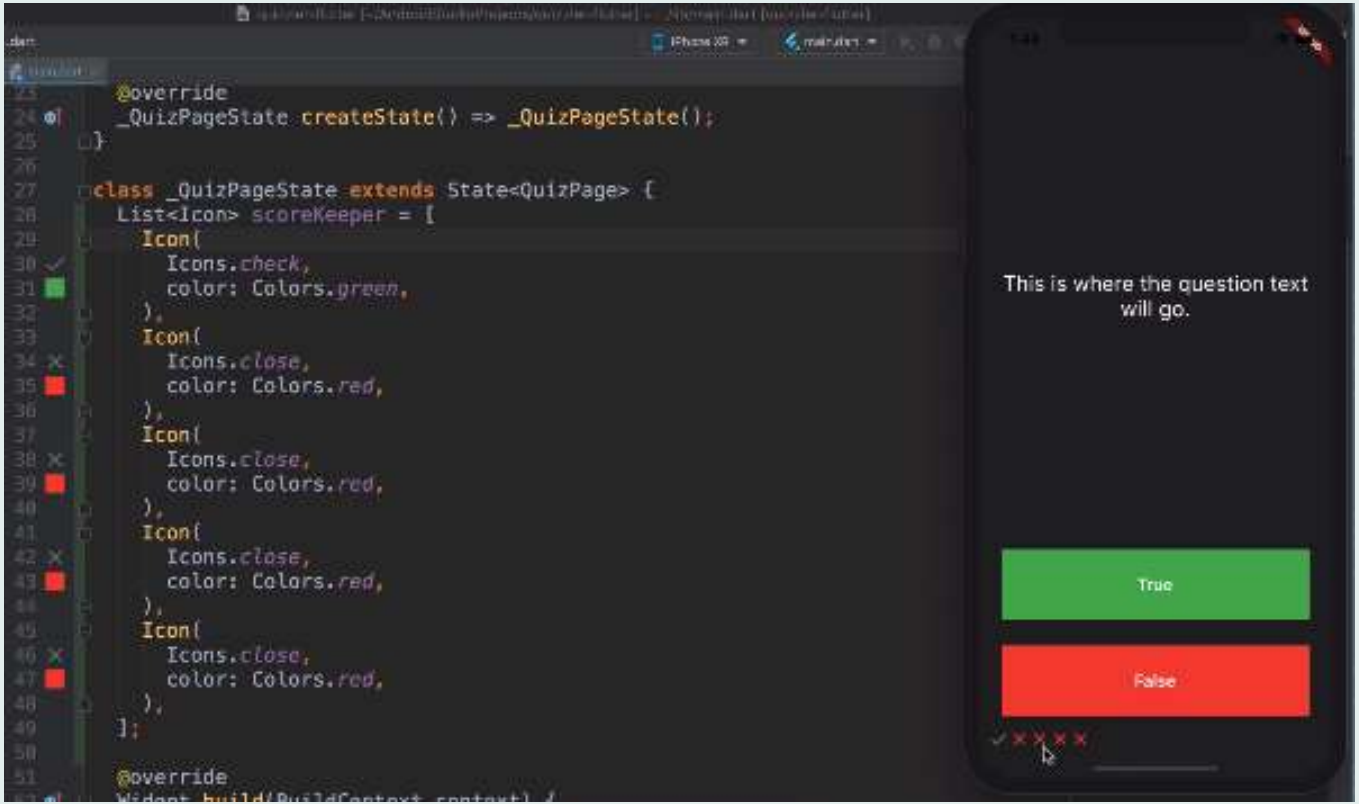
LIST

4. Menghitung jumlah elemen menggunakan properti length

```
List<int> numbers = [1, 2, 3];
print(numbers.length); // Menghitung jumlah elemen dalam list (output: 3)
```

MOJADIPRO

Contoh List



MOJADIPRO

CONDITIONALS

Pernyataan kondisional digunakan untuk membuat keputusan berdasarkan kondisi yang diberikan.
Dart memiliki beberapa bentuk pernyataan kondisional yang umum digunakan, yaitu if, if-else, dan if-else if-else.

MOJADIPRO

Pernyataan if

```
if (condition) {
  // Eksekusi kode jika kondisi bernilai true
}
```

If statement digunakan untuk mengevaluasi suatu kondisi dan menjalankan blok kode jika kondisi tersebut bernilai true.

Contoh :

```
int number = 10;

if (number > 0) {
  print("Number is positive");
}
```

MOJADIPRO

Pernyataan if-else

```
if (condition) {
  // Eksekusi kode jika kondisi bernilai true
} else {
  // Eksekusi kode jika kondisi bernilai false
}
```

MOJADIPRO

Pernyataan if-else

If-else statement digunakan untuk mengevaluasi suatu kondisi, dan menjalankan blok kode dalam if jika kondisi bernilai true, atau menjalankan blok kode dalam else jika kondisi bernilai false.

Contoh :

```
int number = -5;

if (number > 0) {
  print("Number is positive");
} else {
  print("Number is non-positive");
}
```

MOJADIPRO

Pernyataan else-if

```
if (condition1) {
  // Eksekusi kode jika kondisi1 bernilai true
} else if (condition2) {
  // Eksekusi kode jika kondisi1 bernilai
false dan kondisi2 bernil
```

MOJADIPRO

Pernyataan else-if

Else if statement digunakan untuk mengevaluasi beberapa kondisi secara berurutan, dan menjalankan blok kode yang sesuai dengan kondisi pertama yang bernilai true. Jika tidak ada kondisi yang bernilai true, maka akan dijalankan blok kode dalam else (opsional).

```
int number = 0;

if (number > 0) {
  print("Number is positive");
} else if (number < 0) {
  print("Number is negative");
} else {
  print("Number is zero");
}
```

MOJADIPRO

KELAS

Kelas adalah sebuah blueprint atau template yang mendefinisikan struktur dan perilaku objek. Kita dapat menganggap kelas sebagai entitas yang menggambarkan karakteristik dan tindakan objek tertentu.

```
class Mahasiswa {
   String nama;
   String jurusan;
   int semester;

   void perkenalan() {
      print("Halo, nama saya $nama. Saya adalah mahasiswa $jurusan semester $semester.");
   }
}
```

MOJADIPRO

OBJEK

Objek adalah sebuah instansi yang dibuat berdasarkan kelas.
Objek memiliki keadaan (variabel) dan perilaku (metode) yang didefinisikan dalam kelasnya.

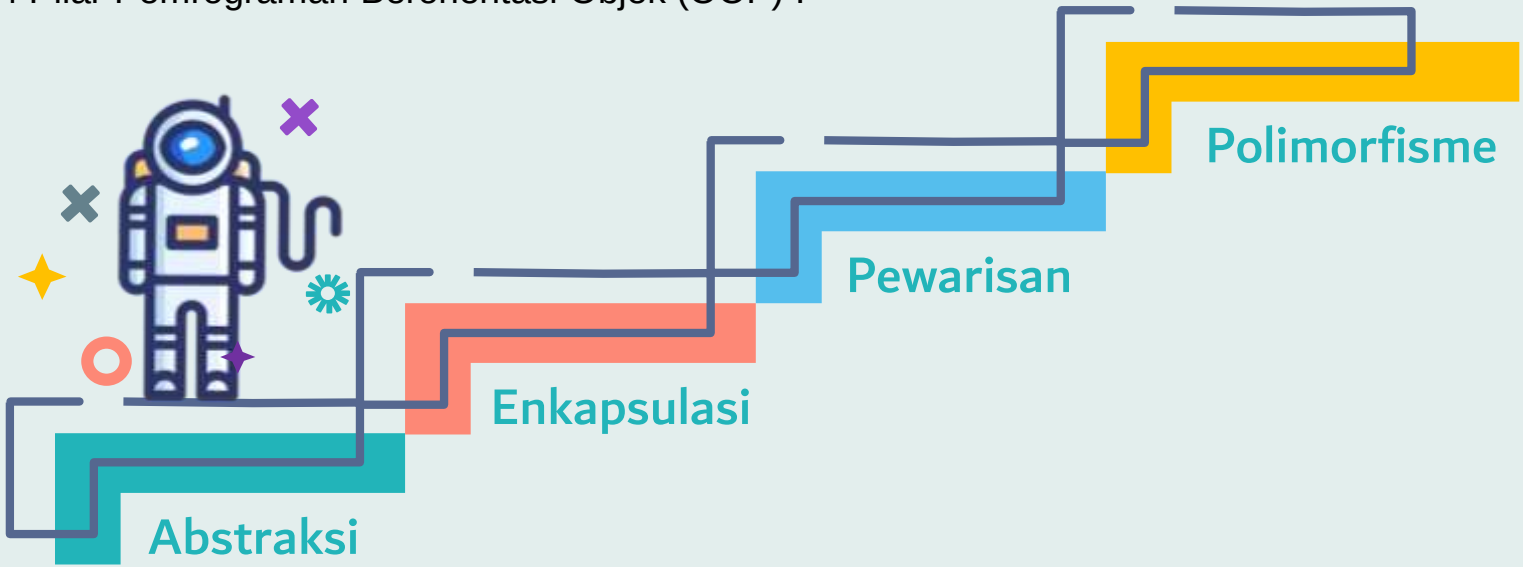
```
void main() {
  Mahasiswa mahasiswa1 = new Mahasiswa();
  mahasiswa1.nama = "John Doe";
  mahasiswa1.jurusan = "Informatika";
  mahasiswa1.semester = 3;
  mahasiswa1.perkenalan(); // Output: "Halo, nama saya John Doe. Saya adalah mahasiswa
Informatika semester 3."

  Mahasiswa mahasiswa2 = new Mahasiswa();
  mahasiswa2.nama = "Jane Smith";
  mahasiswa2.jurusan = "Manajemen";
  mahasiswa2.semester = 2;
  mahasiswa2.perkenalan(); // Output: "Halo, nama saya Jane Smith. Saya adalah mahasiswa
Manajemen semester 2."
}
```

MOJADIPRO

OOP Dart

Dart adalah bahasa pemrograman yang dikembangkan oleh Google. Dart mendukung pemrograman berorientasi objek (Object-Oriented Programming, OOP), yang berarti kita dapat membuat kelas, objek, dan mengelola interaksi antara mereka.

4 Pilar Pemrograman Berorientasi Objek (OOP) :



MOJADIPRO

Abstraksi Pada OOP Dart

Abstraksi memungkinkan Anda untuk menggambarkan objek atau konsep secara umum, tanpa perlu memperhatikan implementasi rinci atau detailnya.
Dalam OOP, abstraksi biasanya dicapai melalui penggunaan kelas abstrak. Kelas abstrak adalah kelas yang tidak dapat diinstansiasi, tetapi dapat diwariskan oleh kelas-kelas turunannya. Kelas abstrak menyediakan kerangka kerja atau kontrak dasar untuk kelas-kelas turunannya, dengan mendefinisikan metode-metode yang harus ada dalam kelas-kelas tersebut.

MOJADIPRO

Konsep Abstraksi Pada OOP Dart

```
abstract class Bentuk {
  void hitungLuas();
  void hitungKeliling();
}

class PersegiPanjang extends Bentuk {
  double panjang, lebar;

  void hitungLuas() {
    double luas = panjang * lebar;
    print('Luas Persegi Panjang: $luas');
  }

  void hitungKeliling() {
    double keliling = 2 * (panjang + lebar);
    print('Keliling Persegi Panjang: $keliling');
  }
}

void main() {
  PersegiPanjang pp = PersegiPanjang();
  pp.panjang = 5;
  pp.lebar = 3;
  pp.hitungLuas();
  pp.hitungKeliling();
}
```

MOJADIPRO

Enkapsulasi Pada OOP Dart

Enkapsulsi adalah salah satu prinsip dalam pemrograman berorientasi objek (OOP) yang menggabungkan data dan metode yang berhubungan dalam sebuah objek dan melindunginya dari akses langsung dari luar.
Dalam bahasa Dart, kita dapat menerapkan enkapsulasi menggunakan konsep pengkapsulan yang melibatkan penggunaan kata kunci public, private, dan protected.

```
class Person {
  String _name; // Variabel privat

  String get name => _name; // Getter untuk mengakses variabel privat

  void set name(String value) {
    if (value.isNotEmpty) {
      _name = value;
    }
  } // Setter untuk mengubah nilai variabel privat
}

void main() {
  Person person = Person();
  person.name = 'John'; // Mengubah nilai variabel privat melalui setter
  print(person.name); // Mengakses nilai variabel privat melalui getter
}
```

MOJADIPRO

Pewarisan Pada OOP Dart

Pewarisan memungkinkan kelas untuk mewarisi atribut dan perilaku dari kelas lain. Dalam Dart, kita dapat menggunakan kata kunci extends untuk mewarisi sebuah kelas.

```
class Student extends Person {
  String school;

  void study() {
    print("$name is studying at $school.");
  }
}

void main() {
  Student student = new Student();
  student.name = "Mike";
  student.age = 20;
  student.school = "ABC University";
  student.sayHello();  // Output: Hello, my name is Mike and I'm 20 years old.
  student.study();     // Output: Mike is studying at ABC University.
}
```

MOJADIPRO

Polimorfisme Pada OOP Dart

Polimorfisme memungkinkan objek untuk mengambil bentuk beberapa tipe. Dalam Dart, polimorfisme dapat dicapai melalui pewarisan dan penggunaan metode yang sama dengan perilaku yang berbeda.

MOJADIPRO

Konsep Polimorfisme Pada OOP Dart

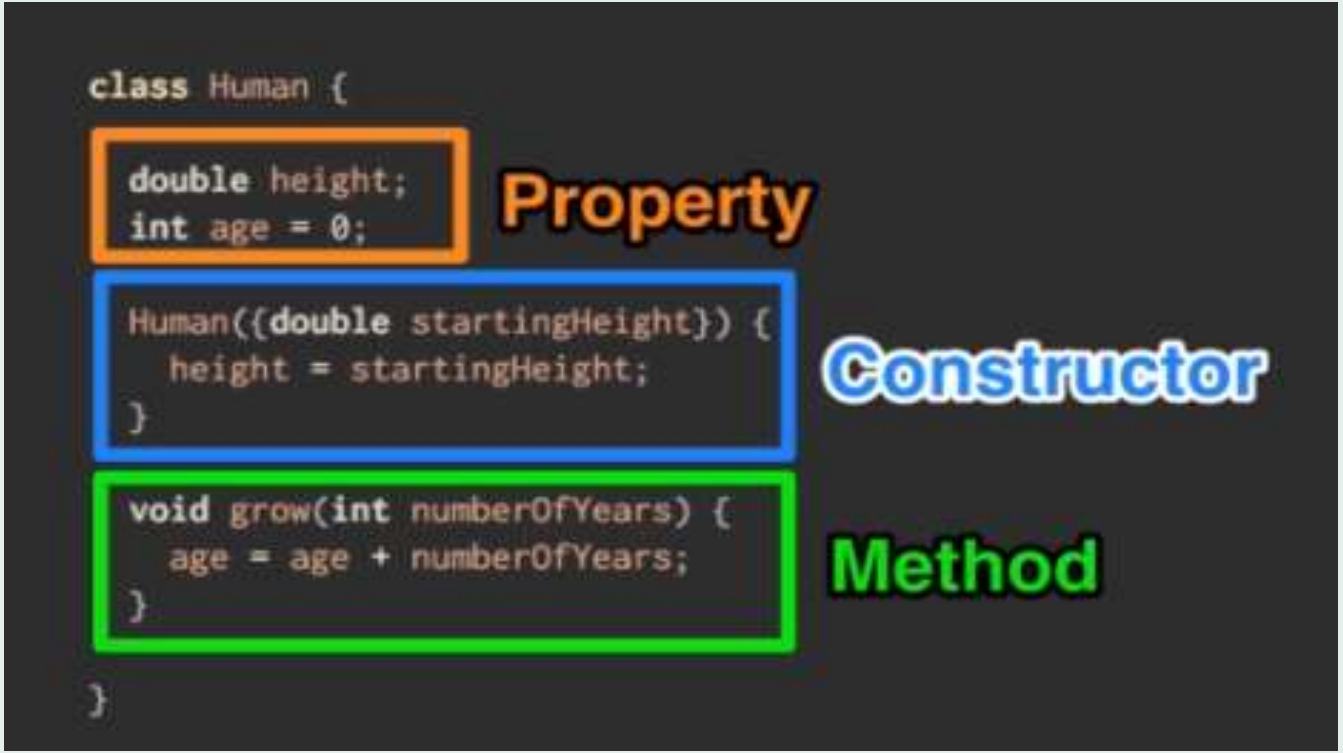
```
class Animal {
  void makeSound() {
    print("The animal makes a sound.");
  }
}

class Cat extends Animal {
  void makeSound() {
    print("The cat meows.");
  }
}

class Dog extends Animal {
  void makeSound() {
    print("The dog barks.");
  }
}

void main() {
  Animal animal = new Animal();
  animal.makeSound();  // Output: The animal makes a sound.

  Cat cat = new Cat();
  cat.makeSound();     // Output: The cat meows.

  Dog dog = new Dog();
  dog.makeSound();     // Output: The dog barks.
}
```

MOJADIPRO

DART CONSTRUCTOR

Constructor adalah metode khusus yang digunakan untuk membuat objek dari sebuah class. Constructor ini dapat digunakan untuk menginisialisasi nilai-nilai awal dari objek tersebut. Dart menyediakan beberapa jenis constructor yang berbeda, yaitu constructor default, constructor bernama, dan constructor dengan parameter.

DART CONSTRUCTOR



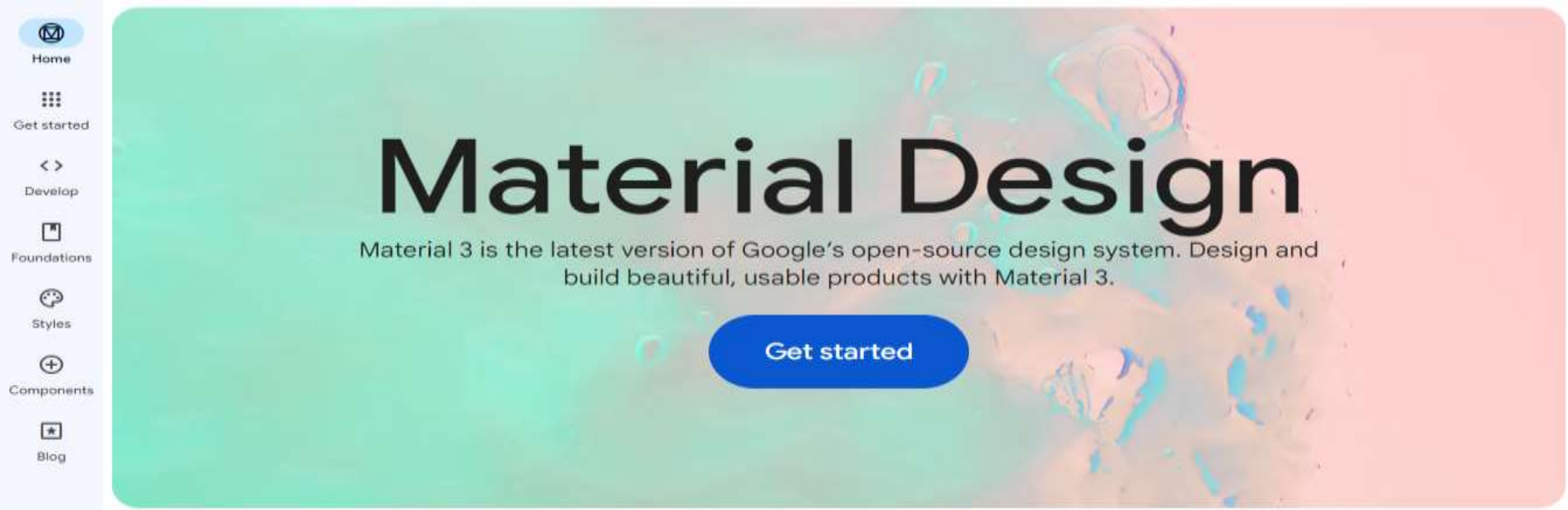
FLUTTER THEMES

Tema Flutter adalah kumpulan gaya dan konfigurasi yang digunakan untuk mengatur tampilan visual aplikasi Flutter.
Dalam Flutter, tema digunakan untuk mengubah aspek-aspek seperti warna, teks, dan gaya widget di seluruh aplikasi dengan cepat dan konsisten.

MOJADIPRO

FLUTTER THEMES

Tema dasar dalam Flutter disebut "Material Design" yang dikembangkan oleh Google. Material Design adalah panduan desain yang mencakup gaya visual, perilaku animasi, dan interaksi pengguna yang konsisten di berbagai platform. Flutter juga mendukung tema kustom yang dapat disesuaikan sesuai kebutuhan aplikasi Anda.



MOJADIPRO

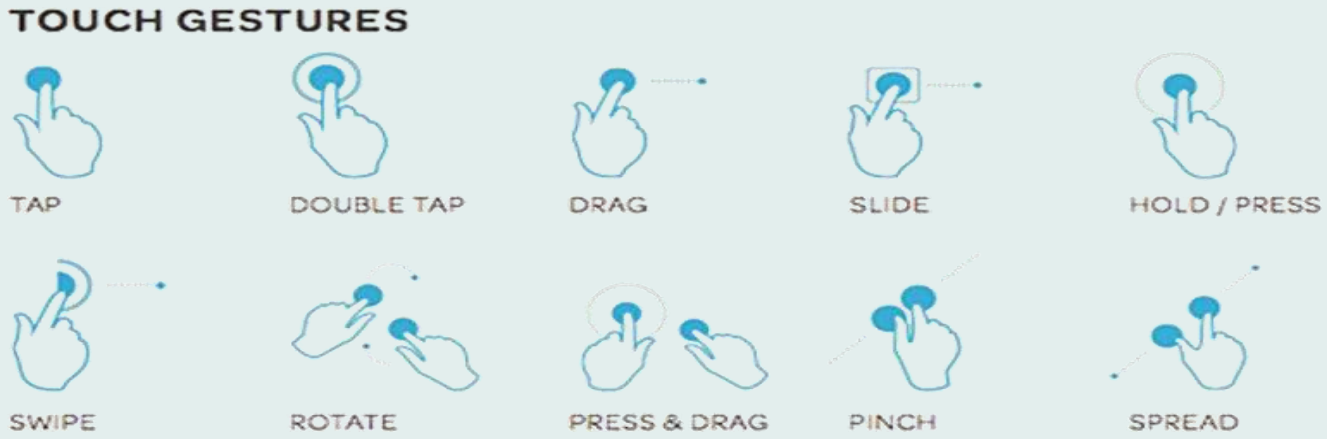
Contoh Penggunaan Flutter Themes

```
MaterialApp(
  title: appName,
  theme: ThemeData(
    // Define the default brightness and colors.
    brightness: Brightness.dark,
    primaryColor: Colors.lightBlue[800],

    // Define the default font family.
    fontFamily: 'Georgia',

    // Define the default `TextTheme`. Use this to specify the default
    // text styling for headlines, titles, bodies of text, and more.
    textTheme: const TextTheme(
      displayLarge: TextStyle(fontSize: 72.0, fontWeight: FontWeight.bold),
      titleLarge: TextStyle(fontSize: 36.0, fontStyle: FontStyle.italic),
      bodyMedium: TextStyle(fontSize: 14.0, fontFamily: 'Hind'),
    ),
  ),
  home: const MyHomePage(
    title: appName,
  ),
);
```

MOJADIPRO

The Gesture Detector Widget

The Gesture Detector widget pada Flutter adalah sebuah widget yang digunakan untuk mendeteksi dan menangani berbagai jenis gesture (gerakan) yang dilakukan pengguna pada layar perangkat. Gesture ini bisa mencakup sentuhan (touch), ketukan (tap), seret (drag), gesekan (swipe), atau gerakan khusus lainnya.



MOJADIPRO

MAPS

Peta (Maps) adalah struktur data yang digunakan untuk memetakan kunci (keys) ke nilai-nilai (values) terkait. Peta juga dikenal dengan sebutan kamus, asosiatif array, atau hash table dalam bahasa pemrograman lainnya.

```
void main() {
  // Membuat peta menggunakan sintaks literal
  var fruits = {'apple': 5, 'banana': 2, 'orange': 10};

  // Menambahkan entri baru ke dalam peta
  fruits['grape'] = 3;

  // Mengakses nilai berdasarkan kunci
  print(fruits['apple']); // Output: 5

  // Menghapus entri berdasarkan kunci
  fruits.remove('banana');

  // Iterasi melalui peta
  fruits.forEach((key, value) {
    print('$key: $value');
  });
}
```

Output :

```
5
apple: 5
orange: 10
grape: 3
```

MOJADIPRO

MAPS

Dart juga menyediakan kelas **Map** yang dapat Anda gunakan untuk membuat peta. Anda dapat mengimpor library **dart:core** dan membuat peta menggunakan konstruktor **Map()**.

```
void main() {
  var map = Map();
  map['a'] = 1;
  map['b'] = 2;
  map['c'] = 3;

  print(map); // Output: {a: 1, b: 2, c: 3}
}
```

MOJADIPRO

ENUMS

Dalam Flutter, enum (singkatan dari enumerations) adalah tipe data yang digunakan untuk mendefinisikan kumpulan nilai yang tetap (konstan). Enumerations memungkinkan Anda untuk menyusun set terbatas dari nilai-nilai yang valid untuk sebuah variabel atau parameter. Anda dapat menggunakan enum untuk menyederhanakan kode, meningkatkan kejelasan, dan mencegah kesalahan dalam penggunaan nilai yang tidak valid.

```
enum EnumName {typeA, typeB, typeC}
```

MOJADIPRO

Contoh Enums

```
enum Status {
  running,
  stopped,
  paused,
}

void main() {
  var status = Status.running;

  if (status == Status.running) {
    print('Program sedang berjalan.');
  } else if (status == Status.stopped) {
    print('Program telah berhenti.');
  } else {
    print('Program dalam status terjeda.');
  }
}
```

MOJADIPRO

TERNARY OPERATOR

Ternary operator memungkinkan Anda mengevaluasi ekspresi secara kondisional dan mengembalikan nilai berdasarkan kondisi yang dihasilkan.

```
(Condition) ? DoThisIfTrue : DoThisIfFalse
```

MOJADIPRO

TERNARY OPERATOR

```
int number = 7;
String result = (number % 2 == 0) ? "Even" : "Odd";
print(result);
```

Pada contoh di atas, jika number adalah bilangan genap, maka result akan berisi "Even". Jika number adalah bilangan ganjil, maka result akan berisi "Odd".

MOJADIPRO

Function as first-class objects

Function as first-class objects (fungsi sebagai objek kelas pertama). Fungsi dapat diberikan sebagai argumen kepada fungsi lain, dikembalikan sebagai nilai dari fungsi lain, dan disimpan dalam variabel.

```
void sayHello() {
  print("Hello!");
}

void main() {
  var greetings = sayHello;
  greetings(); // Memanggil fungsi sayHello melalui variabel greetings
}
```

MOJADIPRO

FINAL

Final digunakan untuk mendeklarasikan variabel yang nilainya tetap setelah nilainya diinisialisasi.

Artinya, setelah nilai variabel final diatur, tidak dapat diubah lagi. Namun, nilai tersebut tidak diketahui secara compile-time dan dapat diinisialisasi pada saat runtime.

MOJADIPRO

FINAL

Contoh kode :

```
final int angka = 10;
```

Pada contoh di atas, variabel angka diinisialisasi dengan nilai 10 dan nilainya tidak dapat diubah setelahnya.

MOJADIPRO

CONST

Const digunakan untuk mendeklarasikan variabel yang nilainya tetap selama waktu kompilasi (compile-time constant).

Ini berarti nilai variabel const harus diketahui pada saat compile-time.

MOJADIPRO

CONST

Contoh kode :

```
const double pi = 3.14;
```

Pada contoh di atas, variabel pi diinisialisasi dengan nilai 3.14 yang diketahui pada saat kompilasi. Karena itu, nilainya tidak dapat diubah.

MOJADIPRO

IMMUTABILITY STATELESS DAN STATEFUL WIDGET

Dalam Flutter, ada dua jenis widget yang berperan dalam membangun antarmuka pengguna :

1. Widget yang tidak dapat diubah (immutability)
2. Widget yang memiliki keadaan (stateless dan stateful widgets).

Stateless Widget: widget yang tidak memiliki keadaan internal yang berubah seiring waktu.
Stateless widget dikenali dengan mengimplementasikan metode build(), yang digunakan untuk menggambar antarmuka pengguna widget berdasarkan input yang diberikan. Ketika ada perubahan pada input, widget ini di-rebuild dengan menggunakan metode build().

Stateful Widget: widget yang memiliki keadaan internal yang dapat berubah selama masa hidupnya. Dalam widget ini, state (keadaan) dapat diperbarui dan dapat mempengaruhi bagaimana antarmuka pengguna ditampilkan. Ketika ada perubahan pada keadaan internal, metode build() akan dipanggil kembali untuk memperbarui tampilan widget.

MOJADIPRO

FLUTTER FAVOURS COMPOSITION VS INHERITANCE

Flutter mengadopsi paradigma pengembangan UI yang disebut "Composition over Inheritance" (Komposisi daripada Pewarisan).
Konsep ini berfokus pada penggunaan komposisi atau penggabungan widget yang ada untuk membangun tampilan UI, daripada mengandalkan pewarisan dari widget yang sudah ada.

MOJADIPRO

Inheritance (Pewarisan):

Dalam pewarisan, sebuah widget dapat mewarisi sifat-sifat dari widget lain dan memperluas fungsionalitasnya. Widget anak memperoleh properti dan metode dari widget induk. Namun, pewarisan memiliki beberapa kelemahan.

MOJADIPRO

Composition (Komposisi):

Dalam komposisi, Anda menggunakan widget yang sudah ada dan menggabungkannya untuk membangun tampilan UI yang kompleks. Anda membuat tampilan UI dengan mengatur widget-widget tersebut secara hierarkis. Setiap widget bertanggung jawab untuk menggambar dirinya sendiri, dan tampilan UI dihasilkan dari gabungan dan penataan widget yang ada.

MOJADIPRO

TERIMA KASIH