

**LAPORAN PRAKTIKUM  
PEMROGRAMAN BERORIENTASI OBYEK**



Disusun Oleh :

Rasya Aditya Amelia Putra (231240001385)

**PRODI TEKNIK INFORMATIKA  
FAKULTAS SAINS DAN TEKNOLOGI  
UNIVERSITAS ISLAM NAHDLATUL ULAMA JEPARA  
TAHUN AKADEMIK 2024/2025**

## DAFTAR ISI

<b>PENDAHULUAN .....</b>	<b>5</b>
• Latar Belakang .....	5
• Tujuan .....	6
<b>ISI PRAKTIKUM .....</b>	<b>8</b>
• <b>Pertemuan 2: Pengenalan Tool dan Instalasi .....</b>	<b>8</b>
• Latihan membuat inputan dengan Dart .....	8
• <b>Pertemuan 3: Perulangan .....</b>	<b>9</b>
• For Loop .....	10
• While Loop .....	12
• Do While Loop .....	13
• Break & Continue .....	14
• For In .....	17
• <b>Pertemuan 4: Stateless dan Stateful Widget .....</b>	<b>18</b>
• Function .....	18
• Function Parameter .....	19
• Optional Parameter .....	20
• Default Value .....	21
• Named Parameter .....	22
• Required Parameter .....	24
• Function Return Value .....	25
• <b>Pertemuan 5: Function Short dan Inner Function .....</b>	<b>26</b>
• Higher Order Function .....	29

• Anonymous Function .....	30
• Scope .....	31
• <b>Pertemuan 6: Classes</b> .....	32
• Dart Classes .....	32
• Getters dan Setters .....	33
• Inheritance .....	34
• Overriding .....	36
• <b>Pertemuan 7: Row and Column</b> .....	38
• Pepak Jawa .....	38
• Cek Bilangan Prima .....	38
• <b>Pertemuan 9: Method Synchronous dan Asynchronous</b> .....	39
• Keyword Future .....	39
• Async dan Await .....	40
• <b>Pertemuan 10-13: Flutter</b> .....	42
• Intro Flutter .....	42
• Navigation dan Hero Animation .....	44
• Form di Flutter .....	47
• <b>Pertemuan 14: SQLite</b> .....	48
• Penjelasan SQLite .....	48
• Implementasi CRUD .....	48
<b>UJIAN AKHIR SEMESTER</b> .....	49
• Demo Aplikasi: Sistem Manajemen Peternakan Sapi .....	49
• Fitur Aplikasi .....	50

<b>Daftar Pustaka .....</b>	<b>52</b>
-----------------------------	-----------

## PENDAHULUAN

### 1. Latar Belakang

Pemrograman Berorientasi Objek (PBO) adalah paradigma pemrograman yang berfokus pada penggunaan objek sebagai entitas yang menggabungkan data dan metode untuk berinteraksi dengan data tersebut. PBO dirancang untuk mempermudah pemeliharaan kode, meningkatkan modularitas, serta memberikan fleksibilitas dalam pengembangan perangkat lunak. Berbeda dengan pendekatan pemrograman prosedural yang menitikberatkan pada fungsi atau prosedur, PBO menempatkan objek sebagai pusat dari proses pengembangan program.

Dalam PBO, dua konsep inti yang mendasarinya adalah kelas dan objek. Kelas berfungsi sebagai cetak biru atau blueprint yang mendefinisikan karakteristik dan perilaku (atribut dan metode) dari objek. Sebagai contoh, dalam sistem manajemen perpustakaan, kelas dapat berupa "Buku" dengan atribut seperti judul, pengarang, dan ISBN, serta metode seperti meminjam atau mengembalikan buku. Objek adalah instansi dari kelas tersebut, yang berarti setiap buku dengan judul dan pengarang tertentu merupakan objek dari kelas "Buku".

PBO memiliki beberapa konsep penting yang mendukung pengembangan perangkat lunak, di antaranya:

- Enkapsulasi: Menggabungkan data dan metode yang berhubungan ke dalam satu unit, sehingga menyembunyikan kompleksitas internal dan hanya menampilkan antarmuka yang diperlukan.
- Pewarisan: Memungkinkan kelas baru mewarisi sifat dan kemampuan dari kelas yang sudah ada, sehingga memfasilitasi penggunaan kembali kode dan mengurangi redundansi.
- Polimorfisme: Memberikan kemampuan kepada objek untuk diproses dengan cara yang berbeda tergantung pada kelas atau tipe datanya, memungkinkan metode yang sama memiliki perilaku yang berbeda.

Dengan konsep-konsep tersebut, PBO menawarkan kerangka kerja yang efisien untuk membangun aplikasi yang kompleks namun tetap mudah dipelihara. PBO juga memungkinkan pemodelan objek dunia nyata atau konsep abstrak dalam kode, memberikan cara intuitif untuk berinteraksi dengan komponen-komponen dalam program.

## 2. Tujuan

1. Mempermudah Pemeliharaan Kode: PBO dirancang untuk membuat kode lebih mudah dipelihara dengan mengorganisasi data dan metode dalam bentuk objek yang modular dan terstruktur.
2. Meningkatkan Modularitas dan Fleksibilitas: Pendekatan berbasis objek memungkinkan pengembang memecah program menjadi komponen-komponen kecil berupa kelas dan objek, sehingga memudahkan pengembangan, pengujian, dan pengelolaan sistem perangkat lunak.
3. Mempermudah Representasi Dunia Nyata: PBO memberikan cara intuitif bagi pemrogram untuk memodelkan objek dunia nyata atau konsep abstrak ke dalam kode, memudahkan pemahaman dan interaksi dengan komponen dalam program.
4. Mengurangi Redundansi Kode: Dengan fitur pewarisan, kelas baru dapat mewarisi atribut dan metode dari kelas yang sudah ada, sehingga mengurangi pengulangan kode dan meningkatkan efisiensi pengembangan.
5. Menyembunyikan Kompleksitas Internal: Melalui konsep enkapsulasi, PBO membantu menyembunyikan detail implementasi internal dan hanya menampilkan antarmuka yang diperlukan, membuat kode lebih aman dan terstruktur.
6. Memfasilitasi Polimorfisme: PBO memungkinkan pengembang menggunakan satu antarmuka untuk memproses berbagai bentuk data, meningkatkan fleksibilitas dalam implementasi metode dan interaksi antara objek yang berbeda.

7. Membangun Aplikasi yang Mudah Dikembangkan dan Dipelihara: Dengan menerapkan prinsip-prinsip seperti enkapsulasi, abstraksi, pewarisan, dan polimorfisme, PBO menyediakan kerangka kerja yang efektif untuk pengembangan perangkat lunak yang kompleks namun tetap modular dan mudah dikelola. Paradigma ini juga membantu menciptakan kode yang lebih bersih, dapat digunakan kembali, dan mudah diperbarui sesuai kebutuhan.

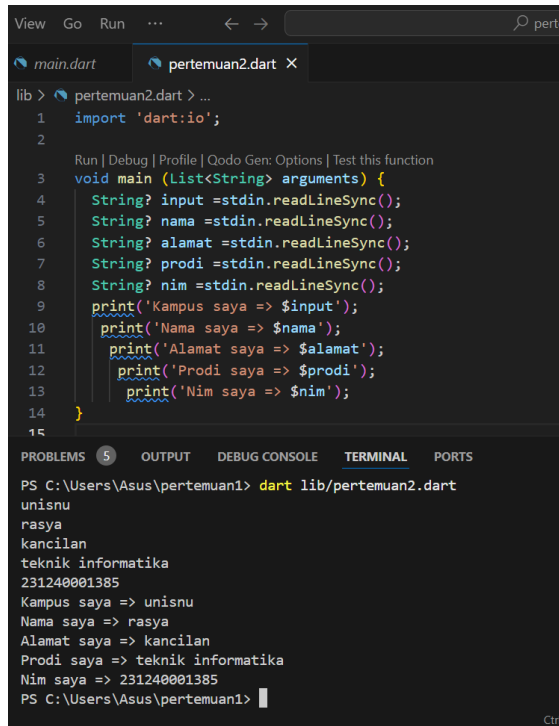
# ISI

## PERTEMUAN 2

### PENGENALAN TOOL DAN INSTALASI

#### 1. Latihan membuat inputan dengan dart

Contoh Penerapan :



```
View Go Run ... < -> per...
main.dart pertemuan2.dart X
lib > pertemuan2.dart > ...
1 import 'dart:io';
2
Run | Debug | Profile | Qodo Gen: Options | Test this function
3 void main (List<String> arguments) {
4   String? input =stdin.readLineSync();
5   String? nama =stdin.readLineSync();
6   String? alamat =stdin.readLineSync();
7   String? prodi =stdin.readLineSync();
8   String? nim =stdin.readLineSync();
9   print('Kampus saya => $input');
10  print('Nama saya => $nama');
11  print('Alamat saya => $alamat');
12  print('Prodi saya => $prodi');
13  print('Nim saya => $nim');
14 }
15

PROBLEMS 5 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Asus\pertemuan1> dart lib/pertemuan2.dart
unisnu
rasya
kancilan
teknik informatika
231240001385
Kampus saya => unisnu
Nama saya => rasya
Alamat saya => kancilan
Prodi saya => teknik informatika
Nim saya => 231240001385
PS C:\Users\Asus\pertemuan1> 
```

Program ini memungkinkan pengguna untuk memasukkan beberapa data melalui terminal, seperti nama kampus, nama pengguna, alamat, program studi, dan NIM. Input data dilakukan menggunakan fungsi `stdin.readLineSync()`, yang membaca data berupa string dari terminal. Data yang dimasukkan kemudian disimpan dalam variabel bertipe nullable seperti `String?` atau dikonversi menjadi tipe data lain, misalnya `int` menggunakan fungsi `int.parse()`. Setelah semua data diinputkan, program mencetak kembali data tersebut ke terminal dengan format yang telah ditentukan menggunakan *string interpolation* (contohnya, `print('Nama Saya => $nama')`). Contoh program ini memberikan gambaran bagaimana cara membuat input dinamis pada aplikasi Dart untuk melatih dasar interaksi pengguna dengan program.



## PERTEMUAN 3

### Perulangan (For Loop)

#### For Loop

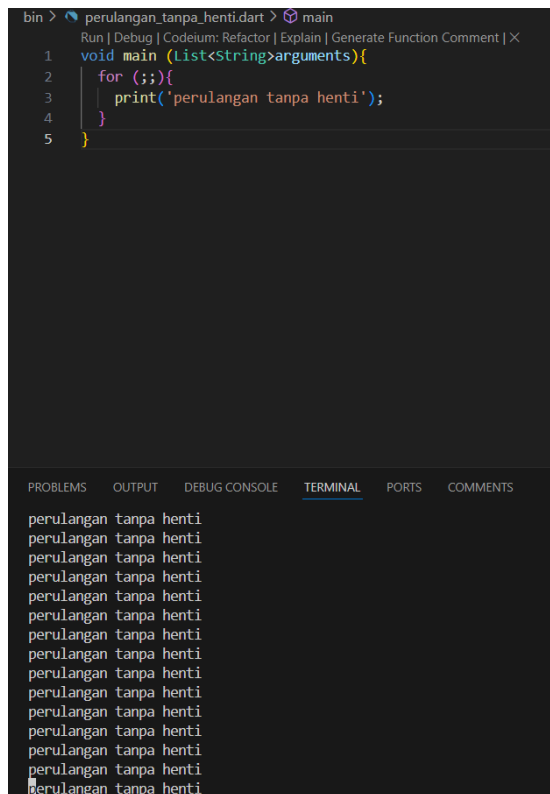
- For adalah salah satu kata kunci yang bisa digunakan untuk melakukan perulangan
- Blok kode yang terdapat di dalam for akan selalu diulangi selama kondisi for terpenuhi

#### Sintak Perulangan For

- Init statement akan dieksekusi hanya sekali di awal sebelum perulangan
- Kondisi akan dilakukan pengecekan dalam setiap perulangan, jika true perulangan akan dilakukan, jika false perulangan akan berhenti
- Post statement akan dieksekusi setiap kali diakhir perulangan
- Init statement, Kondisi dan Post Statement tidak wajib diisi, jika Kondisi tidak diisi, berarti kondisi selalu bernilai true

Contoh Penerapan :

#### 1. Perulangan Tanpa Henti



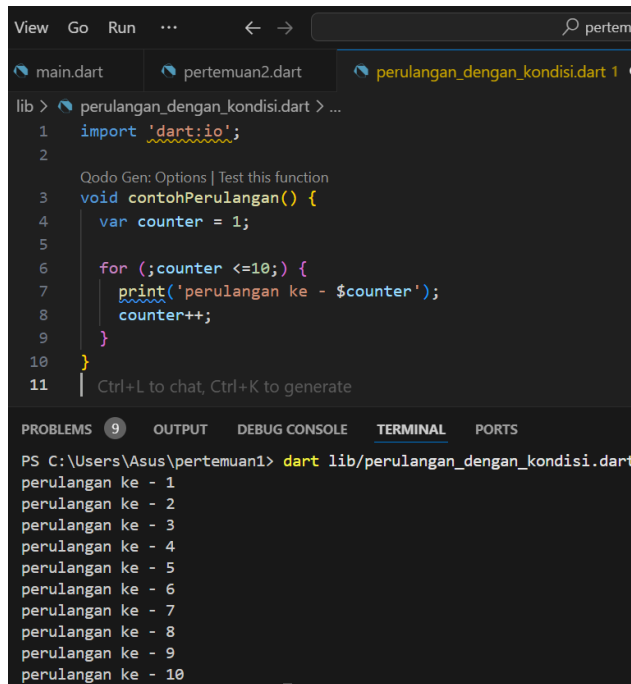
```
bin > perulangan_tanpa_henti.dart > main
Run | Debug | Codeium: Refactor | Explain | Generate Function Comment | X
1 void main (List<String>arguments){
2   for (;;){
3     print('perulangan tanpa henti');
4   }
5 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

```
perulangan tanpa henti
perulangan tanpa henti
perulangan tanpa henti
perulangan tanpa henti
perulangan tanpa henti
perulangan tanpa henti
perulangan tanpa henti
perulangan tanpa henti
perulangan tanpa henti
perulangan tanpa henti
perulangan tanpa henti
perulangan tanpa henti
perulangan tanpa henti
perulangan tanpa henti
perulangan tanpa henti
perulangan tanpa henti
```

Perulangan akan mencetak string 'perulangan tanpa henti' secara terus-menerus ke konsol tanpa henti. Karena tidak ada kondisi keluar dari perulangan, program akan terus berjalan hingga dihentikan secara manual, yang dapat menyebabkan penggunaan CPU yang tinggi.

## 2. Perulangan Dengan Kondisi

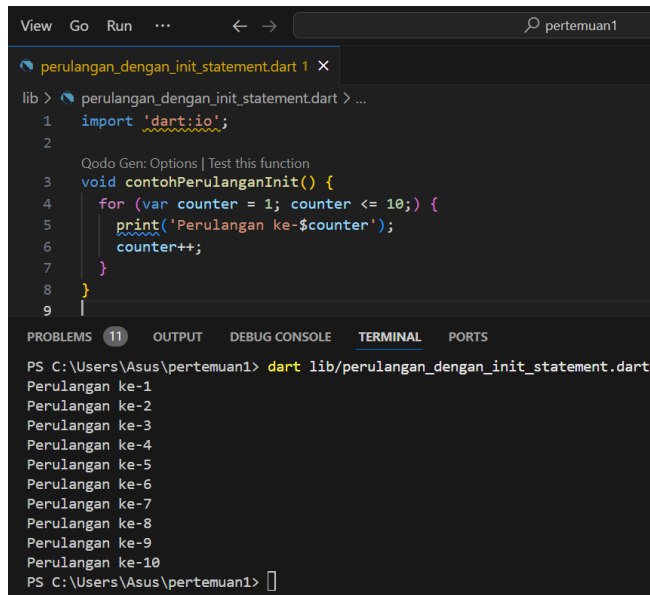


```
View Go Run ... ← → 🔍 pertem
main.dart pertemuan2.dart perulangan_dengan_kondisi.dart 1
lib > perulangan_dengan_kondisi.dart > ...
1 import 'dart:io';
2
3 Qodo Gen: Options | Test this function
4 void contohPerulangan() {
5     var counter = 1;
6
7     for (;counter <=10;) {
8         print('perulangan ke - $counter');
9         counter++;
10    }
11    | Ctrl+L to chat, Ctrl+K to generate

PROBLEMS 9 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Asus\pertemuan1> dart lib/perulangan_dengan_kondisi.dart
perulangan ke - 1
perulangan ke - 2
perulangan ke - 3
perulangan ke - 4
perulangan ke - 5
perulangan ke - 6
perulangan ke - 7
perulangan ke - 8
perulangan ke - 9
perulangan ke - 10
```

Penerapan perulangan dengan kondisi dalam kode Dart memanfaatkan struktur **for loop** tanpa mendefinisikan bagian inisialisasi dan increment secara eksplisit di dalam tanda kurung. Perulangan dimulai dengan nilai awal variabel `counter = 1` yang sudah diinisialisasi sebelum perulangan dimulai. Kondisi pada perulangan adalah `counter <= 10`, yang memastikan bahwa perulangan akan terus berjalan selama nilai `counter` tidak melebihi 10. Pada setiap iterasi, nilai `counter` ditingkatkan dengan menggunakan perintah `counter++`, dan program mencetak pesan "perulangan ke-<angka>".

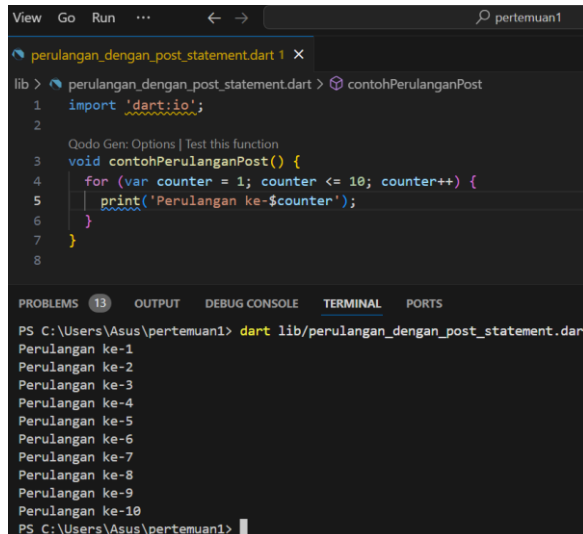
### 3. Perulangan Dengan Init Statement



```
View Go Run ... ← → 🔍 pertemuan1
perulangan_dengan_init_statement.dart 1 X
lib > perulangan_dengan_init_statement.dart > ...
1 import 'dart:io';
2
3 Qodo Gen: Options | Test this function
4 void contohPerulanganInit() {
5   for (var counter = 1; counter <= 10;) {
6     print('Perulangan ke-$counter');
7     counter++;
8   }
9
PROBLEMS 11 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Asus\pertemuan1> dart lib/perulangan_dengan_init_statement.dart
Perulangan ke-1
Perulangan ke-2
Perulangan ke-3
Perulangan ke-4
Perulangan ke-5
Perulangan ke-6
Perulangan ke-7
Perulangan ke-8
Perulangan ke-9
Perulangan ke-10
PS C:\Users\Asus\pertemuan1> 
```

Struktur perulangan ini memiliki tiga bagian utama: init statement, condition, dan increment/decrement statement. Pada kode tersebut, `var counter = 1` adalah bagian init statement yang berfungsi menginisialisasi nilai awal variabel perulangan, yaitu `counter = 1`. Selanjutnya, `counter <= 10` merupakan kondisi yang akan dievaluasi setiap kali perulangan dilakukan, di mana perulangan akan terus berjalan selama nilai `counter` masih kurang dari atau sama dengan 10. Bagian terakhir adalah `counter++`, yang berfungsi untuk menambah nilai `counter` sebesar 1 di setiap iterasi. Hasilnya, program akan mencetak "perulangan ke-1" hingga "perulangan ke-10", menunjukkan proses perulangan yang berjalan sebanyak 10 kali.

#### 4. Perulangan Dengan Post Statement



```
View Go Run ... < -> pertemuan1
perulangan_dengan_post_statement.dart 1 x
lib > perulangan_dengan_post_statement.dart > contohPerulanganPost
1 import 'dart:io';
2
3 Qodo Gen: Options | Test this function
4 void contohPerulanganPost() {
5   for (var counter = 1; counter <= 10; counter++) {
6     print('Perulangan ke-$counter');
7   }
8
9
10 PROBLEMS 13 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Asus\pertemuan1> dart lib/perulangan_dengan_post_statement.dart
Perulangan ke-1
Perulangan ke-2
Perulangan ke-3
Perulangan ke-4
Perulangan ke-5
Perulangan ke-6
Perulangan ke-7
Perulangan ke-8
Perulangan ke-9
Perulangan ke-10
PS C:\Users\Asus\pertemuan1>
```

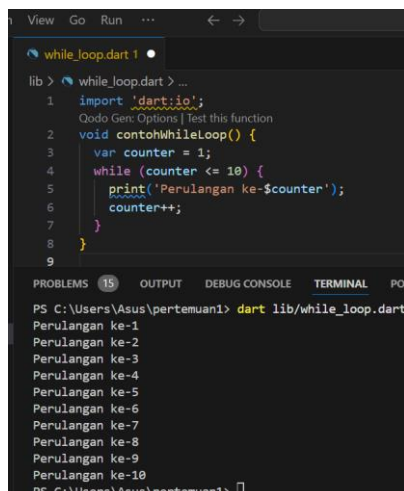
Struktur perulangan for digunakan untuk mencetak teks "perulangan ke-" diikuti angka iterasi dari 1 hingga 10. Post statement pada kode ini terletak pada bagian `counter++`, yang berarti nilai variabel `counter` akan ditambahkan 1 setelah setiap iterasi selesai. Blok perulangan akan terus dijalankan selama kondisi `counter <= 10` bernilai benar.

#### While Loop

While loop adalah versi perulangan yang lebih sederhana dibanding for loop. Di while loop, hanya terdapat kondisi perulangan, tanpa ada init statement dan post statement.

Contoh Penerapan:

- While Loop



```
View Go Run ... < ->
while_loop.dart 1
lib > while_loop.dart > ...
1 import 'dart:io';
2 Qodo Gen: Options | Test this function
3 void contohWhileLoop() {
4   var counter = 1;
5   while (counter <= 10) {
6     print('Perulangan ke-$counter');
7     counter++;
8   }
9
10 PROBLEMS 15 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Asus\pertemuan1> dart lib/while_loop.dart
Perulangan ke-1
Perulangan ke-2
Perulangan ke-3
Perulangan ke-4
Perulangan ke-5
Perulangan ke-6
Perulangan ke-7
Perulangan ke-8
Perulangan ke-9
Perulangan ke-10
PS C:\Users\Asus\pertemuan1>
```

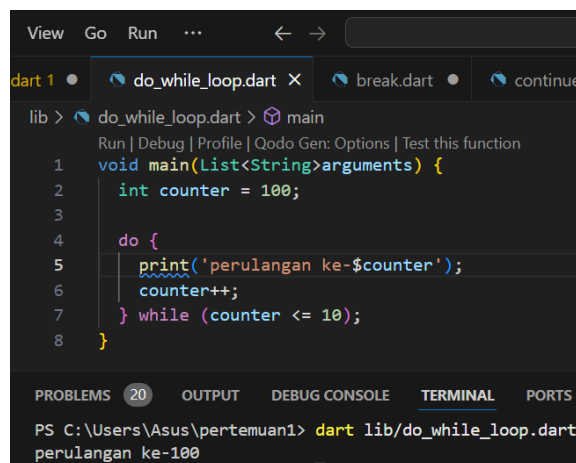
Program dimulai dengan mendeklarasikan variabel counter yang diberi nilai 1. Kemudian, perulangan while akan terus berjalan selama nilai counter kurang dari atau sama dengan 10. Setiap kali perulangan dijalankan, program akan mencetak teks "perulangan ke-<nilai counter>" dan setelah itu nilai counter akan bertambah 1 (counter++). Proses ini akan berulang terus hingga kondisi counter <= 10 tidak lagi terpenuhi, yaitu ketika nilai counter menjadi 11. Dengan demikian, kode ini akan mencetak angka 1 hingga 10.

## Do While Loop

Do While loop adalah perulangan yang mirip dengan while. Perbedaannya hanya pada pengecekan kondisi. Pengecekan kondisi di while loop dilakukan di awal sebelum perulangan dilakukan, sedangkan di do while loop dilakukan setelah perulangan dilakukan. Oleh karena itu dalam do while loop, minimal pasti sekali perulangan dilakukan walaupun kondisi tidak bernilai true.

Contoh Penerapan:

- Do While Loop



```
View Go Run ... < >
dart 1 • do_while_loop.dart X break.dart • continue
lib > do_while_loop.dart > main
Run | Debug | Profile | Qodo Gen: Options | Test this function
1 void main(List<String>arguments) {
2   int counter = 100;
3
4   do {
5     print('perulangan ke-$counter');
6     counter++;
7   } while (counter <= 10);
8 }
PROBLEMS 20 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Asus\pertemuan1> dart lib/do_while_loop.dart
perulangan ke-100
```

Do-while loop adalah jenis perulangan yang menjalankan blok kode setidaknya sekali sebelum memeriksa kondisi untuk melanjutkan perulangan. Dalam contoh ini, variabel counter diinisialisasi dengan nilai 100. Program kemudian mencetak "perulangan ke-100" dan increment counter sebanyak 1 setiap kali perulangan dijalankan. Namun, perulangan berhenti ketika nilai counter lebih besar dari 10.

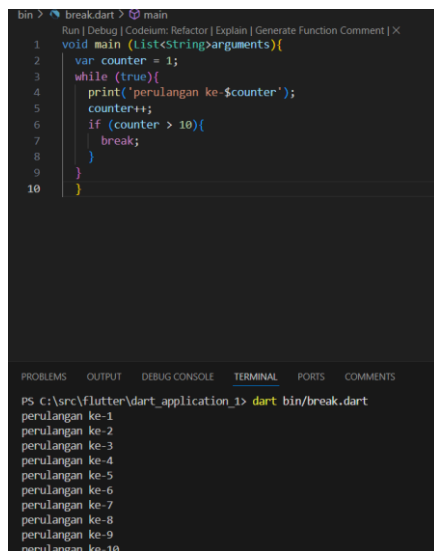
Meskipun kondisi `counter <= 10` tidak pernah terpenuhi karena nilai awalnya 100, perulangan tetap dijalankan setidaknya sekali. Sehingga, output yang dihasilkan hanya satu kali, yaitu "perulangan ke-100", sebelum program selesai.

## Break & Continue

Sama dengan pada perulangan, `break` juga digunakan untuk menghentikan seluruh perulangan. Namun berbeda dengan `continue`, `continue` digunakan untuk menghentikan perulangan saat ini, lalu melanjutkan ke perulangan selanjutnya.

Contoh Penerapan :

### - Break

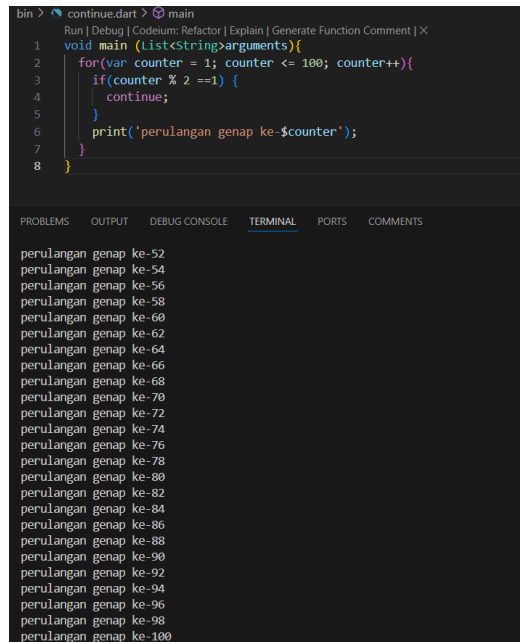


```
bin > break.dart > main
Run | Debug | Codeium: Refactor | Explain | Generate Function Comment | X
1 void main (List<String>arguments){
2   var counter = 1;
3   while (true){
4     print('perulangan ke-$counter');
5     counter++;
6     if (counter > 10){
7       break;
8     }
9   }
10 }

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS
PS C:\src\Flutter\dart_application_1> dart bin/break.dart
perulangan ke-1
perulangan ke-2
perulangan ke-3
perulangan ke-4
perulangan ke-5
perulangan ke-6
perulangan ke-7
perulangan ke-8
perulangan ke-9
perulangan ke-10
```

Penerapan dengan menggunakan `break` menunjukkan cara menghentikan perulangan dalam sebuah struktur `while`. Di dalam loop `while`, perulangan berlangsung tanpa batas karena kondisi `true` selalu bernilai benar. Namun, setiap kali perulangan dijalankan, variabel `counter` ditambahkan satu, dan jika nilai `counter` lebih besar dari 10, perintah `break` akan menghentikan eksekusi perulangan tersebut. Ini berguna untuk mengontrol alur program agar tidak berjalan terus-menerus, sesuai dengan kondisi tertentu yang sudah ditentukan, seperti pada kasus ini saat `counter` mencapai lebih dari 10.

## 5. Continue



```
bin > continue.dart > main
Run | Debug | Codeium: Refactor | Explain | Generate Function Comment | X
1 void main(List<String>arguments){
2   for(var counter = 1; counter <= 100; counter++){
3     if(counter % 2 == 1) {
4       continue;
5     }
6     print('perulangan genap ke-$counter');
7   }
8 }
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS COMMENTS

```
perulangan genap ke-52
perulangan genap ke-54
perulangan genap ke-56
perulangan genap ke-58
perulangan genap ke-60
perulangan genap ke-62
perulangan genap ke-64
perulangan genap ke-66
perulangan genap ke-68
perulangan genap ke-70
perulangan genap ke-72
perulangan genap ke-74
perulangan genap ke-76
perulangan genap ke-78
perulangan genap ke-80
perulangan genap ke-82
perulangan genap ke-84
perulangan genap ke-86
perulangan genap ke-88
perulangan genap ke-90
perulangan genap ke-92
perulangan genap ke-94
perulangan genap ke-96
perulangan genap ke-98
perulangan genap ke-100
```


Continue digunakan dalam sebuah perulangan for untuk melewati nilai tertentu dalam perulangan tersebut. Program ini akan mencetak "perulangan genap ke-..." hanya untuk angka genap dari 1 hingga 100. Setiap kali counter bernilai angka ganjil, kondisi `if(counter % 2 == 1)` akan terpenuhi, sehingga perintah `continue` akan mengeksekusi perulangan selanjutnya tanpa mencetak apa pun untuk angka tersebut. Sebaliknya, ketika counter bernilai genap, angka tersebut akan dicetak dengan pesan yang sesuai. Jadi, `continue` membantu untuk melewati langkah-langkah tertentu dalam sebuah perulangan berdasarkan kondisi yang ditentukan.

## Tanpa For In

Kadang kita biasa mengakses data List menggunakan perulangan. Mengakses data List menggunakan perulangan sangat bertele-tele, kita harus membuat counter, lalu mengakses List menggunakan counter yang kita buat. Namun untungnya, terdapat perulangan for in, yang bisa digunakan untuk mengakses seluruh data di List secara otomatis.

Contoh Penerapan :

- Tanpa For In



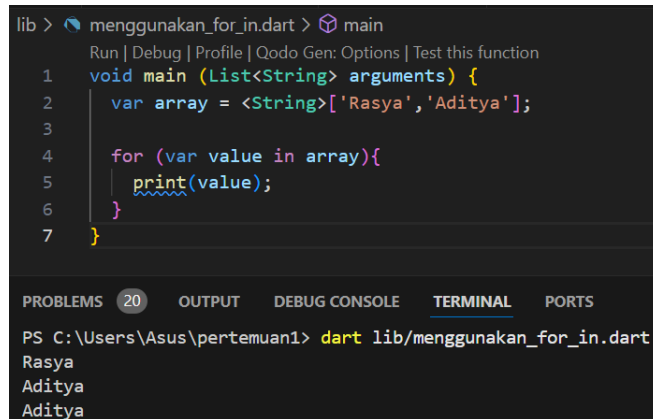
```
lib > tanpa_for_in.dart > contohTanpaForIn
Qodo Gen: Options | Test this function
1 void contohTanpaForIn() {
2   var array = <String>['Rasya', 'Aditya', 'Amelia', 'Putra'];
3
4   for (var i = 0; i < array.length; i++) {
5     print(array[i]);
6   }
7 }
8

PROBLEMS 20 OUTPUT DEBUG CONS Open file in editor (ctrl + click)
PS C:\Users\Asus\pertemuan1> dart lib/tanpa_for_in.dart
Rasya
Aditya
Amelia
Putra
```

Penerapan *loop* tanpa menggunakan kata kunci for-in dapat dilakukan dengan menggunakan struktur for standar seperti pada contoh di atas. Program tersebut mendeklarasikan sebuah *array* bertipe String berisi tiga elemen: 'Rasya', 'Aditya', 'Amelia', dan 'Putra'. Untuk mencetak setiap elemen dalam *array*, digunakan perulangan for dengan indeks yang dimulai dari 0 hingga kurang dari panjang *array* (*array.length*). Pada setiap iterasi, elemen *array* diakses berdasarkan indeksnya (*array[i]*) dan dicetak menggunakan fungsi *print()*. Pendekatan ini memberikan kontrol lebih besar terhadap iterasi, seperti jika diperlukan manipulasi indeks atau logika tambahan di dalam perulangan.



- Menggunakan For In



```
lib > menggunakan_for_in.dart > main
Run | Debug | Profile | Qodo Gen: Options | Test this function
1 void main (List<String> arguments) {
2   var array = <String>['Rasya','Aditya'];
3
4   for (var value in array){
5     print(value);
6   }
7 }
```

PROBLEMS 20 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\Asus\pertemuan1> dart lib/menggunakan_for_in.dart
Rasya
Aditya
Aditya
```

Penerapan for in digunakan untuk mengiterasi elemen-elemen di dalam sebuah list atau array. List yang dideklarasikan dengan nama array berisi tiga elemen string: 'Rasya' dan 'Aditya'. Melalui perulangan for in, setiap elemen dalam list tersebut diakses satu per satu dan disimpan sementara dalam variabel value. Kemudian, elemen tersebut dicetak ke layar menggunakan perintah `print(value)`. Dengan pendekatan ini, for in memungkinkan iterasi yang lebih sederhana dan efisien tanpa perlu menggunakan indeks untuk mengakses elemen array, sehingga cocok untuk kasus di mana hanya elemen array yang diperlukan tanpa manipulasi berdasarkan indeksnya. Hasil dari kode ini adalah mencetak setiap elemen array ke layar secara berurutan: "Rasya", "Aditya".

## PERTEMUAN 4

### Stateles dan Stateful Widget

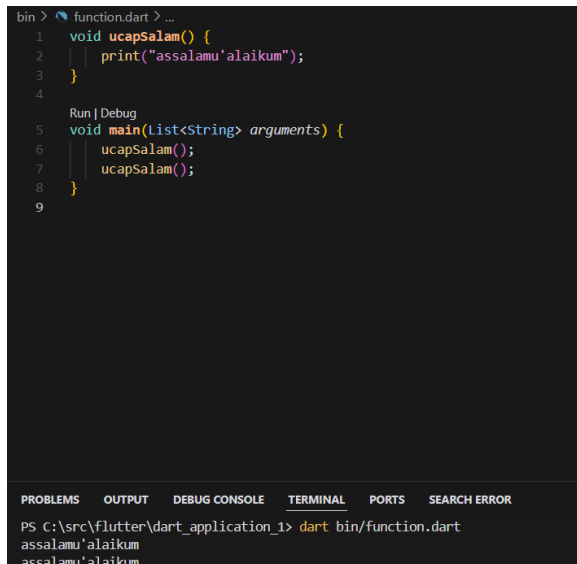
#### Function

Function adalah blok kode program yang akan berjalan saat kita panggil. Sebelumnya kita sudah menggunakan method `print()` untuk menampilkan tulisan di console. Untuk membuat function, kita bisa menggunakan kata kunci `void`, lalu diikuti dengan nama function, kurung `()` dan diakhiri dengan block.

Kita bisa memanggil function dengan menggunakan nama function lalu diikuti dengan kurung `()`. Di bahasa pemrograman lain, Function juga disebut dengan Method.

Contoh Penerapan :

- Function.dart



```
bin > function.dart > ...
1 void ucapSalam() {
2   | print("assalamu'alaikum");
3 }
4
Run | Debug
5 void main(List<String> arguments) {
6   | ucapSalam();
7   | ucapSalam();
8 }
9

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SEARCH ERROR
PS C:\src\flutter\dart_application_1> dart bin/function.dart
assalamu'alaikum
assalamu'alaikum
```

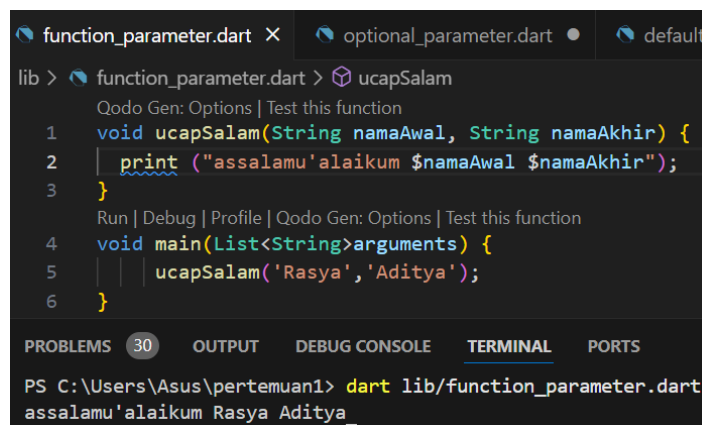
Function pada bahasa pemrograman Dart diterapkan untuk membuat kode lebih modular dan reusable. Fungsi `ucapSalam` adalah sebuah fungsi tanpa parameter yang didefinisikan untuk mencetak teks "assalamu'alaikum" ke konsol. Fungsi ini dipanggil dua kali di dalam fungsi utama `main`, yang merupakan titik awal eksekusi program. Dengan menggunakan fungsi seperti ini, kode menjadi lebih terstruktur, memudahkan pengembang untuk mengelola dan menghindari pengulangan kode.

## Function Parameter

Kita bisa mengirim informasi ke function yang ingin kita panggil. Untuk melakukan hal tersebut, kita perlu menambahkan parameter atau argument di function yang sudah kita buat. Cara membuat parameter sama seperti cara membuat variabel. Parameter ditempatkan di dalam kurung () di deklarasi function Parameter bisa lebih dari satu, jika lebih dari satu, harus dipisah menggunakan tanda koma. Ketika memanggil function, kita bisa sebut nama function nya, lalu gunakan kurung (), jika terdapat parameter dalam kurung (), silahkan masukkan parameter sesuai dengan jumlah parameter nya.

Contoh Penerapannya :

- Function Parameter.dart



```
function_parameter.dart X optional_parameter.dart • default_
lib > function_parameter.dart > ucapSalam
Qodo Gen: Options | Test this function
1 void ucapSalam(String namaAwal, String namaAkhir) {
2   print ("assalamu'alaikum $namaAwal $namaAkhir");
3 }
Run | Debug | Profile | Qodo Gen: Options | Test this function
4 void main(List<String>arguments) {
5   ucapSalam('Rasya', 'Aditya');
6 }
PROBLEMS 30 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Asus\pertemuan1> dart lib/function_parameter.dart
assalamu'alaikum Rasya Aditya
```

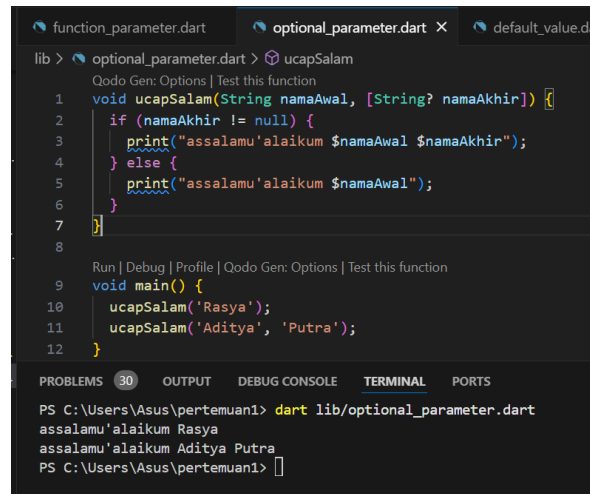
Fungsi ucapSalam didefinisikan dengan dua parameter bertipe String, yaitu namaAwal dan namaAkhir, yang berfungsi sebagai input untuk menyusun pesan sapaan. Dalam fungsi main, ucapSalam dipanggil dengan argumen 'Rasya' dan 'Aditya', sehingga nilai 'Rasya' diberikan ke parameter namaAwal dan 'Saputra' ke parameter namaAkhir. Fungsi tersebut kemudian memproses input tersebut untuk mencetak output personalisasi berupa "assalamu'alaikum Rasya Aditya". Contoh ini menunjukkan bagaimana parameter pada fungsi digunakan untuk mengolah data masukan menjadi output yang sesuai kebutuhan, sehingga mendukung modularitas dan efisiensi kode.

## Optional Parameter

Secara default, parameter wajib dikirim ketika kita memanggil function. Namun jika kita ingin membuat parameter yang optional, artinya tidak wajib dikirim, kita bisa wrap dalam kurung []. Dan parameter optional haruslah nullable.

Contoh Penerapan :

- Optional Parameter.dart



```
lib > optional_parameter.dart > ucapSalam
Qodo Gen: Options | Test this function
1 void ucapSalam(String namaAwal, [String? namaAkhir]) {
2   if (namaAkhir != null) {
3     print("assalamu'alaikum $namaAwal $namaAkhir");
4   } else {
5     print("assalamu'alaikum $namaAwal");
6   }
7 }
8
Run | Debug | Profile | Qodo Gen: Options | Test this function
9 void main() {
10  ucapSalam('Rasya');
11  ucapSalam('Aditya', 'Putra');
12 }

PROBLEMS 30 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Asus\pertemuan1> dart lib/optional_parameter.dart
assalamu'alaikum Rasya
assalamu'alaikum Aditya Putra
PS C:\Users\Asus\pertemuan1>
```

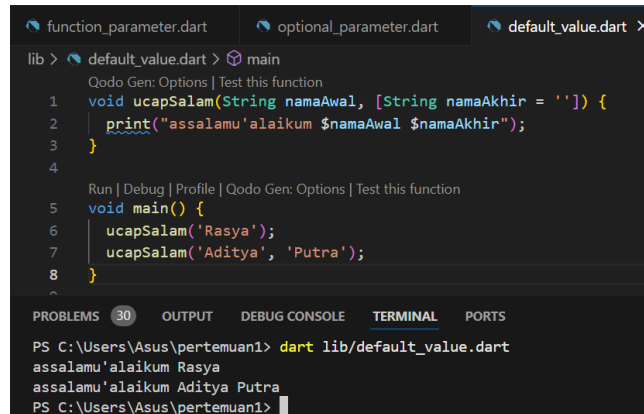
Fungsi `ucapSalam` memiliki dua parameter, yaitu `namaAwal` sebagai parameter wajib dan `[String? namaAkhir]` sebagai parameter opsional yang ditandai dengan tanda kurung siku (`[]`) dan tipe data yang dapat bernilai null menggunakan simbol `?`. Saat fungsi dipanggil di dalam `main`, parameter opsional `namaAkhir` dapat diisi ataupun tidak. Jika parameter opsional tidak diisi, nilainya akan otomatis menjadi null. Dalam implementasi, pemanggilan `ucapSalam('Rasya')` hanya menggunakan `namaAwal`, sehingga outputnya adalah "assalamu'alaikum Rasya null". Sedangkan pemanggilan `ucapSalam('Aditya', 'Putra')` menggunakan kedua parameter, sehingga menghasilkan output "assalamu'alaikum Aditya Putra". Hal ini menunjukkan fleksibilitas fungsi dalam menangani argumen berbeda.

## Default Value

Jika optional parameter tidak ingin nullable, maka kita wajib menambahkan default value. Caranya, setelah nama parameter, kita tambahkan = default value.

Contoh Penerapan :

- Default Value.dart



```
function_parameter.dart optional_parameter.dart default_value.dart X
lib > default_value.dart > main
Qodo Gen: Options | Test this function
1 void ucapSalam(String namaAwal, [String namaAkhir = '']) {
2   print("assalamu'alaikum $namaAwal $namaAkhir");
3 }
4
Run | Debug | Profile | Qodo Gen: Options | Test this function
5 void main() {
6   ucapSalam('Rasya');
7   ucapSalam('Aditya', 'Putra');
8 }
PROBLEMS 30 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Asus\pertemuan1> dart lib/default_value.dart
assalamu'alaikum Rasya
assalamu'alaikum Aditya Putra
PS C:\Users\Asus\pertemuan1>
```

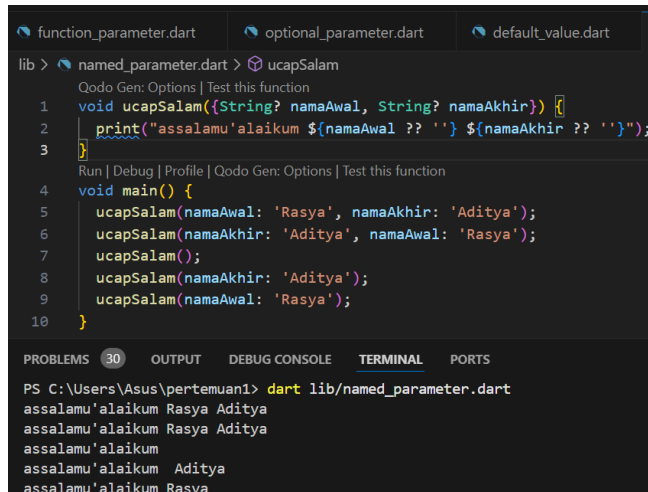
Penerapan default value menunjukkan fungsi ucapSalam memiliki dua parameter: namaAwal, yang bersifat wajib, dan namaAkhir, yang bersifat opsional dengan nilai default berupa string kosong (""). Jika parameter namaAkhir tidak diberikan saat pemanggilan fungsi, maka nilai default tersebut akan digunakan. Pada fungsi main, fungsi ucapSalam dipanggil dua kali: pertama dengan hanya memberikan parameter namaAwal bernilai 'Rasya', sehingga menghasilkan output "assalamu'alaikum Rasya ", dan kedua dengan memberikan kedua parameter, yaitu namaAwal bernilai 'Aditya' dan namaAkhir bernilai 'Putra', sehingga menghasilkan output "assalamu'alaikum Aditya Putra". Penerapan ini memudahkan fleksibilitas fungsi dalam menerima argumen tanpa memaksa semua parameter untuk selalu diisi.

## Named Parameter

Jika optional parameter tidak ingin nullable, maka kita wajib menambahkan default value. Caranya, setelah nama parameter, kita tambahkan = default value.

Contoh Penerapan :

- Named Parameter.dart



```
function_parameter.dart optional_parameter.dart default_value.dart
lib > named_parameter.dart > ucapSalam
Qodo Gen: Options | Test this function
1 void ucapSalam({String? namaAwal, String? namaAkhir}) {
2   print("assalamu'alaikum ${namaAwal ?? ''} ${namaAkhir ?? ''}");
3 }
Run | Debug | Profile | Qodo Gen: Options | Test this function
4 void main() {
5   ucapSalam(namaAwal: 'Rasya', namaAkhir: 'Aditya');
6   ucapSalam(namaAkhir: 'Aditya', namaAwal: 'Rasya');
7   ucapSalam();
8   ucapSalam(namaAkhir: 'Aditya');
9   ucapSalam(namaAwal: 'Rasya');
10 }

PROBLEMS 30 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Asus\pertemuan1> dart lib/named_parameter.dart
assalamu'alaikum Rasya Aditya
assalamu'alaikum Rasya Aditya
assalamu'alaikum
assalamu'alaikum Aditya
assalamu'alaikum Rasya
```

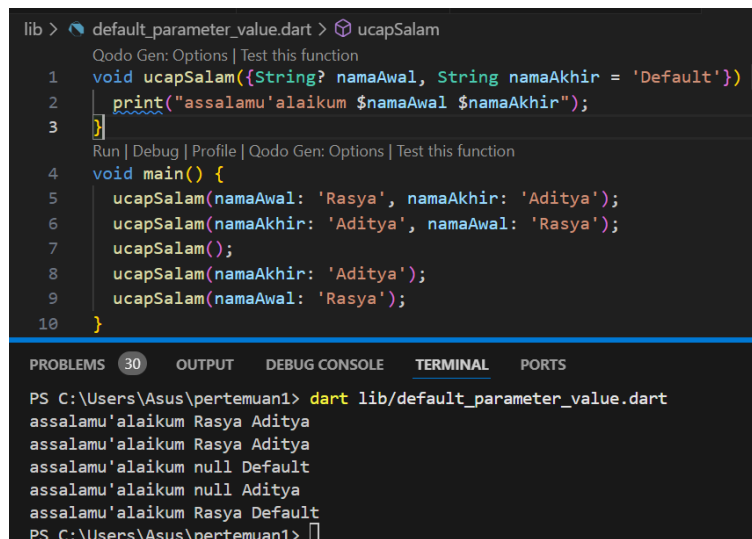
Penerapan named parameter menunjukkan fungsi ucapSalam yang menggunakan named parameters {String? namaAwal, String? namaAkhir} untuk mencetak salam dengan menyisipkan nama awal dan nama akhir. Named parameters memungkinkan argumen dilewatkan tanpa mengikuti urutan tertentu, seperti terlihat pada pemanggilan ucapSalam(namaAwal: 'Rasya', namaAkhir: 'Aditya') dan ucapSalam(namaAkhir: 'Aditya', namaAwal: 'Rasya'), yang keduanya menghasilkan output yang sama. Parameter juga bersifat opsional karena diberi tipe nullable (String?), sehingga fungsi tetap dapat dipanggil tanpa parameter, seperti ucapSalam(), atau hanya dengan salah satu parameter, seperti ucapSalam(namaAwal: 'Rasya'). Penerapan ini meningkatkan keterbacaan kode dan mempermudah penggunaan fungsi dalam berbagai skenario.

## Default Parameter Value

Karena secara default, named parameter adalah nullable, sehingga secara otomatis ketika memanggil function, kita tidak wajib mengirim parameter tersebut. Agar nilai parameter tidak null, kita juga bisa memberikan default value dengan menambah = nilai default nya.

Contoh penerapan :

- Default Parameter Value.dart



```
lib > default_parameter_value.dart > ucapSalam
Qodo Gen: Options | Test this function
1 void ucapSalam({String? namaAwal, String namaAkhir = 'Default'}) {
2   print("assalamu'alaikum $namaAwal $namaAkhir");
3 }
Run | Debug | Profile | Qodo Gen: Options | Test this function
4 void main() {
5   ucapSalam(namaAwal: 'Rasya', namaAkhir: 'Aditya');
6   ucapSalam(namaAkhir: 'Aditya', namaAwal: 'Rasya');
7   ucapSalam();
8   ucapSalam(namaAkhir: 'Aditya');
9   ucapSalam(namaAwal: 'Rasya');
10 }

PROBLEMS 30 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Asus\pertemuan1> dart lib/default_parameter_value.dart
assalamu'alaikum Rasya Aditya
assalamu'alaikum Rasya Aditya
assalamu'alaikum null Default
assalamu'alaikum null Aditya
assalamu'alaikum Rasya Default
PS C:\Users\Asus\pertemuan1>
```

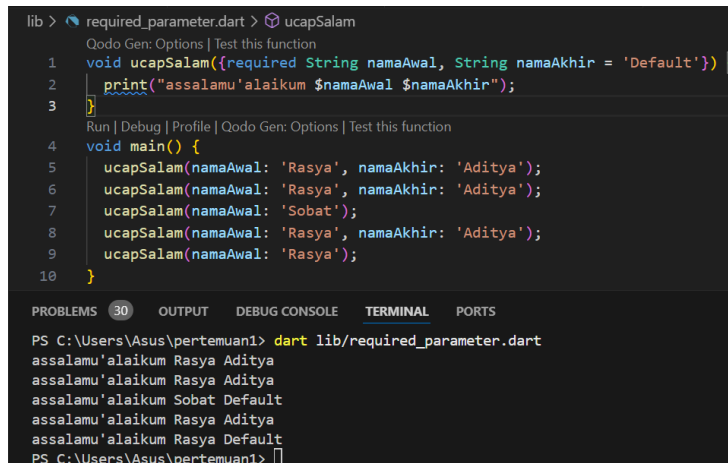
Penerapan materi default parameter value menunjukkan penggunaan parameter opsional dengan nilai default pada fungsi ucapSalam. Parameter namaAkhir memiliki nilai default 'Default', sehingga jika tidak diberikan nilai saat fungsi dipanggil, parameter tersebut akan menggunakan nilai default tersebut. Dengan memanfaatkan parameter opsional dan default value, fungsi dapat lebih fleksibel karena dapat dipanggil dengan berbagai kombinasi argumen. Dalam fungsi main(), terdapat beberapa cara pemanggilan fungsi ucapSalam, baik dengan memberikan kedua parameter, hanya salah satu parameter, atau bahkan tanpa parameter. Jika namaAwal atau namaAkhir tidak diberikan, maka nilai default akan digunakan, menghasilkan keluaran seperti assalamu'alaikum null Default untuk pemanggilan tanpa parameter sama sekali.

## Required Parameter

Pada named parameter, kita juga bisa memaksa sebuah parameter menjadi mandatory, sehingga kita memanggil function tersebut, parameter nya wajib ditambahkan. Caranya kita bisa tambahkan kata kunci required di awal parameter.

Contoh Penerapan :

- Required Parameter.dart



```
lib > required_parameter.dart > ucapSalam
Qodo Gen: Options | Test this function
1 void ucapSalam({required String namaAwal, String namaAkhir = 'Default'}) {
2   print("assalamu'alaikum $namaAwal $namaAkhir");
3 }
Run | Debug | Profile | Qodo Gen: Options | Test this function
4 void main() {
5   ucapSalam(namaAwal: 'Rasya', namaAkhir: 'Aditya');
6   ucapSalam(namaAwal: 'Rasya', namaAkhir: 'Aditya');
7   ucapSalam(namaAwal: 'Sobat');
8   ucapSalam(namaAwal: 'Rasya', namaAkhir: 'Aditya');
9   ucapSalam(namaAwal: 'Rasya');
10 }

PROBLEMS 30 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Asus\pertemuan1> dart lib/required_parameter.dart
assalamu'alaikum Rasya Aditya
assalamu'alaikum Rasya Aditya
assalamu'alaikum Sobat Default
assalamu'alaikum Rasya Aditya
assalamu'alaikum Rasya Default
PS C:\Users\Asus\pertemuan1>
```

Penerapan required parameter menunjukkan fungsi ucapSalam memiliki parameter opsional namaAkhir dengan nilai default 'Default' dan parameter wajib namaAwal yang ditandai dengan keyword required. Fungsi ini mencetak ucapan salam yang disesuaikan berdasarkan parameter yang diberikan. Pada bagian main, pemanggilan fungsi ucapSalam menunjukkan fleksibilitas penerapan parameter opsional dan parameter wajib. Jika namaAwal tidak diberikan, kode akan menghasilkan error karena parameter tersebut wajib diisi, sedangkan namaAkhir akan menggunakan nilai default jika tidak disediakan.



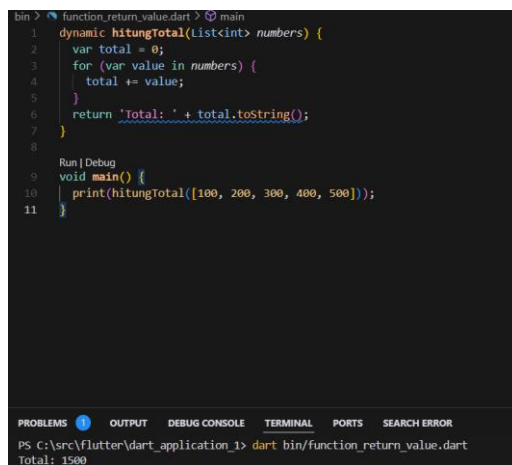
## Function Return Value

Secara default, function itu menghasilkan value null, namun jika kita ingin, kita bisa membuat sebuah function yang mengembalikan nilai.

Agar function bisa menghasilkan value, kita harus mengubah kata kunci void dengan tipe data yang dihasilkan. Dan di dalam block function, untuk menghasilkan nilai tersebut, kita harus menggunakan kata kunci return, lalu diikuti dengan data yang sesuai dengan tipe data yang sudah kita deklarasikan di function.

Kita hanya bisa menghasilkan 1 data di sebuah function, tidak bisa lebih dari satu.

Contoh Penerapan :



```
bin > function_return_value.dart > main
1 dynamic hitungTotal(List<int> numbers) {
2   var total = 0;
3   for (var value in numbers) {
4     total += value;
5   }
6   return 'Total: ' + total.toString();
7 }
8
Run | Debug
9 void main() {
10  print(hitungTotal([100, 200, 300, 400, 500]));
11 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SEARCH ERROR

PS C:\src\flutter\dart\_application\_1> dart bin/function\_return\_value.dart  
Total: 1500

Penerapan function return value menunjukkan fungsi hitungTotal dideklarasikan dengan tipe data dynamic untuk fleksibilitas tipe data yang dikembalikan. Fungsi ini menerima parameter berupa daftar bilangan integer (List<int> numbers), menjumlahkan seluruh elemen dalam daftar menggunakan perulangan for, dan menyimpan hasil penjumlahan dalam variabel total. Hasil akhirnya dikonversi menjadi string dan digabungkan dengan teks "total" sebelum dikembalikan oleh fungsi. Pada fungsi main, daftar angka [100, 200, 300, 400, 500] diberikan sebagai argumen untuk fungsi hitungTotal, dan hasilnya ditampilkan menggunakan perintah print, menghasilkan keluaran berupa teks: total 1500.

## PERTEMUAN 5

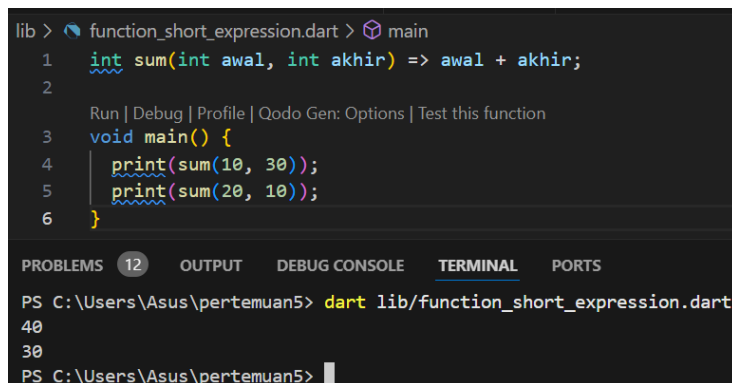
### Function Short

#### Function Short Expression

Dart mendukung function short expression. Dimana jika terdapat sebuah function yang hanya satu baris, kita bisa menyingkatnya secara sederhana. Untuk membuat function short expression, kita tidak butuh kurung `{}` dan juga tidak butuh kata kunci `return`.

Contoh Penerapan :

- Function Short Expression.dart



```
lib > function_short_expression.dart > main
1  int sum(int awal, int akhir) => awal + akhir;
2
Run | Debug | Profile | Qodo Gen: Options | Test this function
3  void main() {
4    print(sum(10, 30));
5    print(sum(20, 10));
6  }

PROBLEMS 12 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Asus\pertemuan5> dart lib/function_short_expression.dart
40
30
PS C:\Users\Asus\pertemuan5>
```

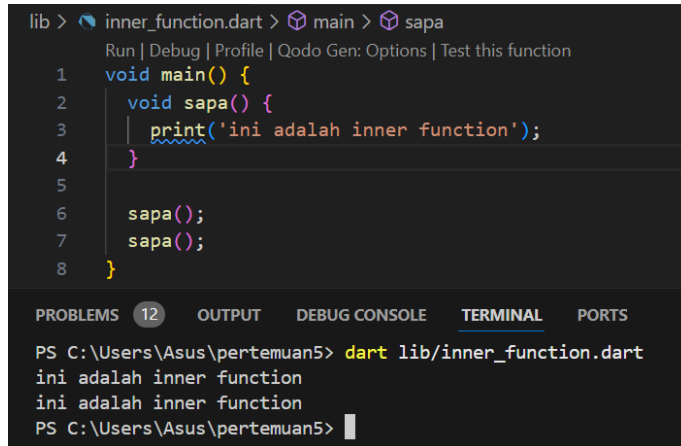
Function Short Expression di Dart adalah cara singkat untuk mendeklarasikan fungsi yang hanya memiliki satu pernyataan (ekspresi). Fungsi `sum` digunakan untuk menjumlahkan dua bilangan, yaitu `awal` dan `akhir`. Fungsi ini ditulis dalam bentuk pendek: `int sum(int awal, int akhir) => awal + akhir;`, di mana `awal + akhir` adalah ekspresi yang langsung dikembalikan sebagai hasil. Dalam program utama (`main()`), fungsi ini dipanggil dengan parameter berbeda, misalnya `sum(10, 30)` menghasilkan 40 dan `sum(20, 10)` menghasilkan 30.

#### Inner Function

Di Dart, kita bisa membuat inner function di dalam outer function. Namun perlu diperhatikan, inner function yang dibuat di dalam outer function, hanya bisa diakses dari outer function saja, tidak bisa diakses dari luar outer function. Untuk lebih detail tentang ini akan kita bahas di materi tentang Scope.

Contoh Penerapan :

- Inner Function.dart



```
lib > inner_function.dart > main > sapa
Run | Debug | Profile | Qodo Gen: Options | Test this function
1 void main() {
2   void sapa() {
3     print('ini adalah inner function');
4   }
5
6   sapa();
7   sapa();
8 }

PROBLEMS 12 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Asus\pertemuan5> dart lib/inner_function.dart
ini adalah inner function
ini adalah inner function
PS C:\Users\Asus\pertemuan5>
```

konsep *inner function* dalam bahasa Dart diterapkan dengan mendefinisikan fungsi sapa di dalam fungsi main. Fungsi sapa hanya dapat diakses dari dalam fungsi main karena berada dalam lingkup (*scope*) lokalnya. Ketika sapa() dipanggil, program akan menjalankan isi fungsi tersebut, yaitu mencetak tulisan "ini adalah inner function" ke konsol. Pemanggilan sapa() sebanyak dua kali menunjukkan bahwa fungsi ini dapat digunakan berulang-ulang selama berada dalam lingkup yang sama.

## Main Function

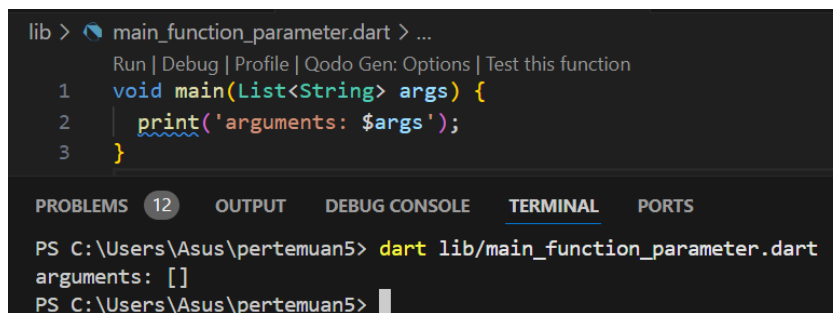
Main function adalah function yang digunakan sebagai gerbang masuk aplikasi Dart. Function main adalah function yang akan dijalankan pertama kali oleh Dart.

## Main Function Parameter

Main function memiliki sebuah parameter optional, yaitu adalah arguments, dimana data parameter tersebut berupa List<String>.

Contoh Penerapan :

- Main Function Parameter.dart

A screenshot of an IDE interface. The top part shows a code editor with a Dart file named 'main\_function\_parameter.dart'. The code defines a 'main' function that takes a 'List<String> args' parameter and prints it. The bottom part shows a terminal window with the command 'dart lib/main\_function\_parameter.dart' executed, resulting in the output 'arguments: []'.

```
lib > main_function_parameter.dart > ...  
Run | Debug | Profile | Qodo Gen: Options | Test this function  
1 void main(List<String> args) {  
2   print('arguments: $args');  
3 }  
  
PROBLEMS 12 OUTPUT DEBUG CONSOLE TERMINAL PORTS  
PS C:\Users\Asus\pertemuan5> dart lib/main_function_parameter.dart  
arguments: []  
PS C:\Users\Asus\pertemuan5>
```

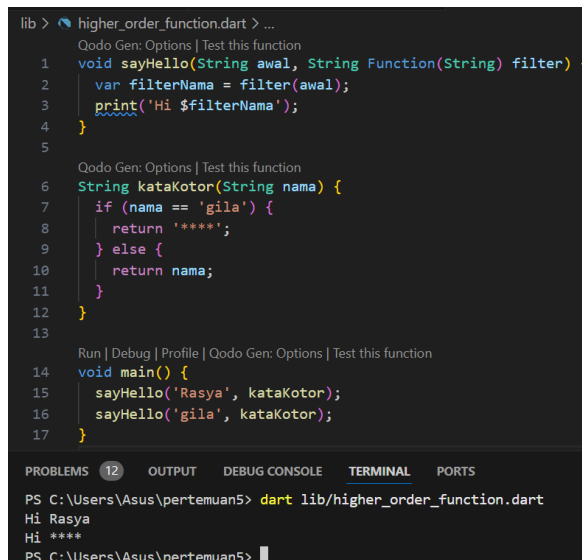
Fungsi main dideklarasikan dengan parameter berupa sebuah list string (List<String> args), yang memungkinkan program menerima input dari command line. Setiap argumen yang dimasukkan setelah nama file akan dimasukkan ke dalam list args. Kemudian, program mencetak daftar argumen tersebut ke output menggunakan perintah print.

## Higher Order Function

Higher Order Function adalah function yang menggunakan function sebagai variable, parameter atau return value. Penggunaan Higher-Order Function kadang berguna ketika kita ingin membuat function yang general dan ingin mendapatkan input yang flexible berupa function, yang bisa dideklarasikan oleh pengguna ketika memanggil function tersebut.

Contoh Penerapan :

- Higher Order Function.dart



```
lib > higher_order_function.dart > ...
Qodo Gen: Options | Test this function
1 void sayHello(String awal, String Function(String) filter) {
2   var filterNama = filter(awal);
3   print('Hi $filterNama');
4 }
5
Qodo Gen: Options | Test this function
6 String kataKotor(String nama) {
7   if (nama == 'gila') {
8     return '****';
9   } else {
10    return nama;
11  }
12 }
13
Run | Debug | Profile | Qodo Gen: Options | Test this function
14 void main() {
15   sayHello('Rasya', kataKotor);
16   sayHello('gila', kataKotor);
17 }

PROBLEMS 12 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Asus\pertemuan5> dart lib/higher_order_function.dart
Hi Rasya
Hi ****
PS C:\Users\Asus\pertemuan5>
```

Fungsi sayHello adalah Higher Order Function karena menerima parameter filter, yaitu fungsi lain dengan tipe String Function(String). Fungsi ini digunakan untuk memproses input sebelum menampilkan hasilnya. Di sini, fungsi kataKotor berfungsi sebagai filter untuk mengganti kata tertentu (seperti "gila") menjadi sensor berupa "\*\*\*\*". Saat fungsi sayHello dipanggil, input nama akan diproses oleh fungsi kataKotor terlebih dahulu sebelum ditampilkan.

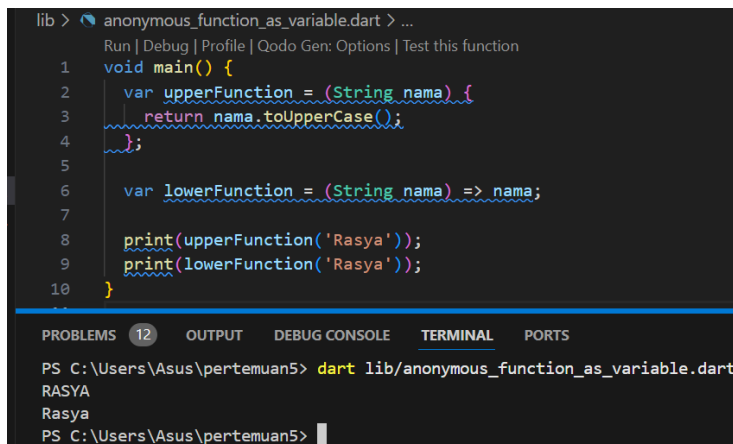
## Anonymous Function

Pembuatan anonymous function mirip seperti function bisanya, namun yang membedakan adalah tidak ada nama function nya.

Biasanya anonymous function sering digunakan ketika memanggil function yang membutuhkan parameter berupa function.

Contoh Penerapan :

### - Anonymous Function



```
lib > anonymous_function_as_variable.dart > ...
Run | Debug | Profile | Qodo Gen: Options | Test this function
1 void main() {
2   var upperFunction = (String nama) {
3     return nama.toUpperCase();
4   };
5
6   var lowerFunction = (String nama) => nama;
7
8   print(upperFunction('Rasya'));
9   print(lowerFunction('Rasya'));
10 }

PROBLEMS 12 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Asus\pertemuan5> dart lib/anonymous_function_as_variable.dart
RASYA
Rasya
PS C:\Users\Asus\pertemuan5>
```

Pada penerapan tersebut terdapat dua fungsi anonim: upperFunction dan lowerFunction. Fungsi upperFunction menggunakan sintaks biasa untuk mengubah teks menjadi huruf besar dengan memanfaatkan metode bawaan .toUpperCase(). Sementara itu, lowerFunction ditulis menggunakan arrow function (=>) untuk mengembalikan nilai string tanpa memodifikasinya. Ketika upperFunction('Rasya') dipanggil, hasilnya adalah "RASYA", sedangkan lowerFunction('Rasya') mengembalikan string "Rasya".

## Scope

Scope adalah Variable atau Function hanya bisa diakses di dalam area dimana mereka dibuat.

Contoh Penerapan :

### - Scope.dart

```
lib > scope.dart > main
Run | Debug | Profile | Qodo Gen: Options | Test this function
1 void main() {
2   var nama = 'Rasya';
3   String ucapSalam() {
4     var salam = "assalamu'alaikum $nama";
5     return salam;
6   }
7   Ctrl+L to chat, Ctrl+K to generate
8   var salam = ucapSalam();
9   print(salam);
10 }

PROBLEMS 12 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Asus\pertemuan5> dart lib/scope.dart
assalamu'alaikum Rasya
PS C:\Users\Asus\pertemuan5>
```

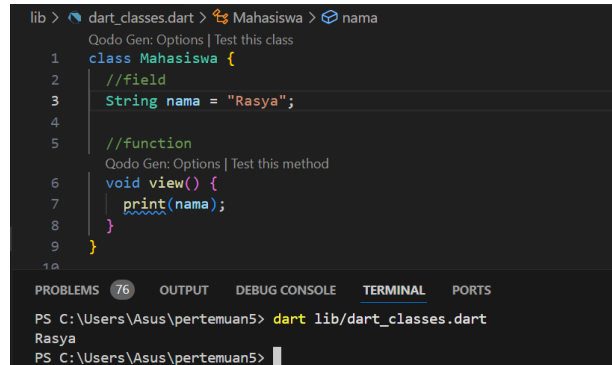
```
lib > scope2.dart > main
Run | Debug | Profile | Qodo Gen: Options | Test this function
1 void main() {
2   var nama = 'Rasya';
3
4   String ucapSalam() {
5     var salam = "assalamu'alaikum $nama";
6     print(salam);
7   }
8   ucapSalam();
9   print(salam);
10 }
--
PROBLEMS 15 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Asus\pertemuan5> dart lib/scope2.dart
assalamu'alaikum Rasya
PS C:\Users\Asus\pertemuan5> dart lib/scope2.dart
lib/scope2.dart:10:9: Error: Undefined name 'salam'.
  print(salam);
  ~~~~~
lib/scope2.dart:4:3: Error: A non-null value must be returned since the return type 'String'
'doesn't allow null'.
  String ucapSalam() {
  ~~~~~
```

Dalam penerapan tersebut, variabel nama dideklarasikan di tingkat global sehingga dapat diakses di seluruh fungsi dalam program. Sedangkan variabel salam dideklarasikan di dalam fungsi ucapSalam(), sehingga cakupan (scope) variabel ini hanya terbatas di dalam fungsi tersebut. Ketika kita mencoba mengakses variabel salam di luar fungsi, program menghasilkan error karena variabel tersebut tidak dikenali di luar lingkup (scope) tempat ia dideklarasikan.

## PERTEMUAN 6

### Classes

#### 1. Dart Classes



```
lib > dart_classes.dart > Mahasiswa > nama
Qodo Gen: Options | Test this class
1 class Mahasiswa {
2   //field
3   String nama = "Rasya";
4
5   //function
6   Qodo Gen: Options | Test this method
7   void view() {
8     print(nama);
9   }
10 }
PROBLEMS 76 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Asus\pertemuan5> dart lib/dart_classes.dart
Rasya
PS C:\Users\Asus\pertemuan5>
```

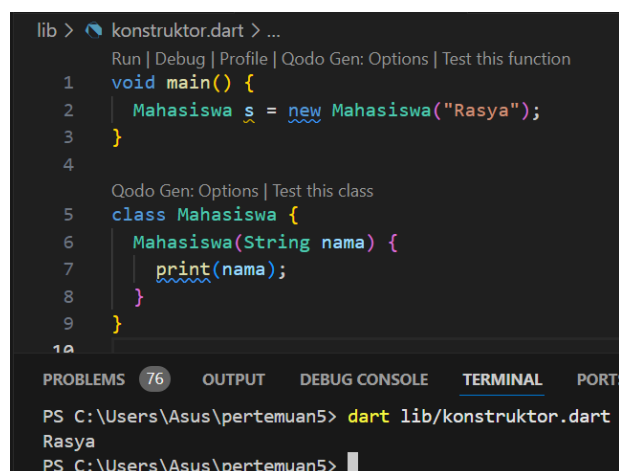
Contoh di atas mendeklarasikan kelas Siswa. Kelas tersebut memiliki sebuah nama. view() adalah fungsi sederhana yang mencetak nilai dari nama.

#### Konstruktor Dart

Konstruktor dalam bahasa pemrograman dart adalah fungsi khusus dari suatu kelas yang bertanggung jawab untuk menginisialisasi variabel yang ada di suatu kelas.

Contoh Penerapan :

- Konstruktor.dart



```
lib > konstruktor.dart > ...
Run | Debug | Profile | Qodo Gen: Options | Test this function
1 void main() {
2   Mahasiswa s = new Mahasiswa("Rasya");
3 }
4
Qodo Gen: Options | Test this class
5 class Mahasiswa {
6   Mahasiswa(String nama) {
7     print(nama);
8   }
9 }
10 }
PROBLEMS 76 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Asus\pertemuan5> dart lib/konstruktor.dart
Rasya
PS C:\Users\Asus\pertemuan5>
```



Penerapan Konstruktor menunjukkan kelas Mahasiswa memiliki konstruktor bernama Mahasiswa yang menerima parameter String nama. Ketika objek s dari kelas Mahasiswa dibuat menggunakan new Mahasiswa("Rasya"), konstruktor ini secara otomatis dipanggil, sehingga nilai "Rasya" dikirim ke parameter nama, dan fungsi print(nama) menampilkan teks "Rasya".

## Getters dan Setters

Getters dan Setters juga disebut sebagai ‘Accessors’ dan ‘Mutators’ yang memungkinkan program untuk menginisialisasi dan mengambil nilai masing-masing bidang kelas. Getter atau pengakses didefinisikan menggunakan kata kunci get. Setter atau mutator didefinisikan menggunakan kata kunci set.

### Contoh Penerapan :

- Getter dan Setter.dart

```
lib > getter_setter.dart > Mahasiswa
Oodo Gen: Options | Test this class
1 class Mahasiswa {
2   late String nama;
3   late int umur;
4   Mahasiswa(this.nama, this.umur);
5   String get nama_mahasiswa {
6     return nama;
7   }
8   void set nama_mahasiswa(String nama) {
9     this.nama = nama;
10  }
11  void set umur_mahasiswa(int umur) {
12    if (umur <= 0) {
13      print("Umur harus lebih besar dari 5");
14    } else {
15      this.umur = umur;
16    }
17  }
18  int get umur_mahasiswa {
19    return umur;
20  }
21 }
Run | Debug | Profile | Qodo Gen: Options | Test this function
22 void main() {
23   Mahasiswa s1 = new Mahasiswa('Amirul', 10);
24   print(s1.nama_mahasiswa);
25   print(s1.umur_mahasiswa);
26 }
27
```

```
PS C:\Users\Asus\pertemuan5> dart lib/getter_setter.dart
Amirul
10
```

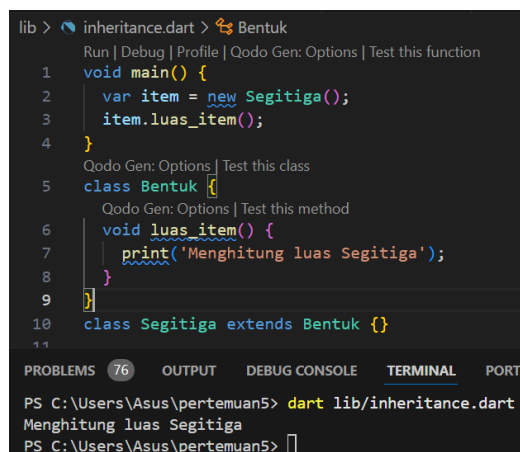
Penerapan tersebut menunjukkan kelas Mahasiswa memiliki atribut nama dan umur. Getter (get) digunakan untuk mengambil nilai dari atribut, seperti nama\_mahasiswa dan umur\_mahasiswa, yang mengembalikan nilai nama dan umur. Setter (set) digunakan untuk mengatur nilai atribut dengan kontrol tertentu. Misalnya, pada setter umur\_mahasiswa, terdapat validasi bahwa umur tidak boleh kurang dari atau sama dengan 0, sehingga menjaga konsistensi data. Penerapan ini memungkinkan pengembang membatasi akses langsung ke atribut, memastikan data diolah sesuai aturan yang ditentukan. Dalam contoh, ketika objek s1 dibuat, nilai nama dan umur dapat diakses melalui getter dan diubah dengan setter jika diperlukan.

## Inheritance

Inheritance merupakan kemampuan dari suatu program untuk membuat kelas baru dari kelas yang ada. Kelas tersebut adalah kelas yang diperluas untuk membuat kelas yang lebih baru atau dalam dart dikenal dengan istilah parent class atau super class. Sementara kelas yang baru dibuat disebut dengan child class atau subclass.

Contoh Penerapan :

- Inheritance.dart



```
lib > inheritance.dart > Bentuk
Run | Debug | Profile | Qodo Gen: Options | Test this function
1 void main() {
2   var item = new Segitiga();
3   item.luas_item();
4 }
Qodo Gen: Options | Test this class
5 class Bentuk {
6   void luas_item() {
7     print('Menghitung luas Segitiga');
8   }
9 }
10 class Segitiga extends Bentuk {}
11

PROBLEMS 76 OUTPUT DEBUG CONSOLE TERMINAL PORT
PS C:\Users\Asus\pertemuan5> dart lib/inheritance.dart
Menghitung luas Segitiga
PS C:\Users\Asus\pertemuan5>
```

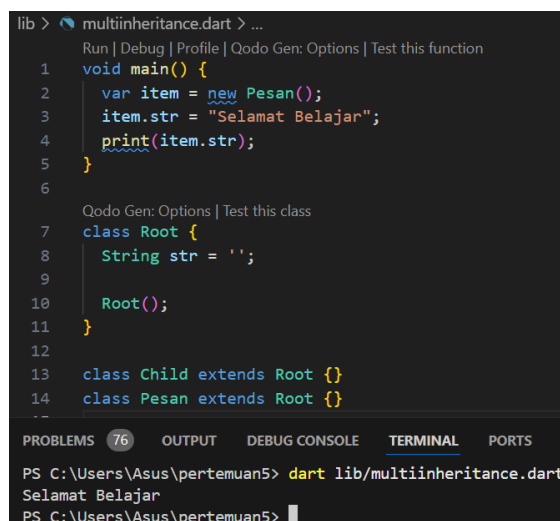
Kelas Segitiga mewarisi semua properti dan metode dari kelas induknya, yaitu Bentuk, menggunakan keyword `extends`. Hal ini berarti, meskipun kelas Segitiga tidak mendeklarasikan metode `luas_item()`, kelas tersebut tetap bisa menggunakan metode tersebut karena sudah diwariskan dari kelas Bentuk. Ketika objek Segitiga dibuat di dalam fungsi `main`, metode `luas_item()` dipanggil melalui objek `item`, dan program akan mencetak "Menghitung luas Segitiga" sesuai definisi dalam kelas induk.

Jenis – jenis inheritance

- Single Setiap class paling banyak dapat diperpanjang dari satu parent class.
- Multiple Sebuah kelas dapat mewarisi dari beberapa kelas. Dart tidak mendukung pewarisan berganda.
- Multi-level Sebuah kelas dapat mewarisi dari kelas anak lain.

Contoh Penerapan :

- Multiinheritance.dart



```
lib > multiinheritance.dart > ...
Run | Debug | Profile | Qodo Gen: Options | Test this function
1 void main() {
2   var item = new Pesan();
3   item.str = "Selamat Belajar";
4   print(item.str);
5 }
6
Qodo Gen: Options | Test this class
7 class Root {
8   String str = '';
9
10  Root();
11 }
12
13 class Child extends Root {}
14 class Pesan extends Root {}
--
PROBLEMS 76 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Asus\pertemuan5> dart lib/multiinheritance.dart
Selamat Belajar
PS C:\Users\Asus\pertemuan5>
```

Penerapan tersebut terdapat sebuah kelas induk bernama `Root` yang memiliki properti `str` bertipe `String` dan dua kelas turunan yaitu `Child` dan `Pesan` yang mewarisi sifat dari kelas `Root`. Ketika program dijalankan melalui fungsi `main()`, sebuah objek `item` dibuat dari kelas `Pesan`, kemudian nilai "Selamat Belajar" diassign ke properti `str` yang diwarisi dari kelas `Root`, dan akhirnya nilai tersebut dicetak.

## Overriding

Overriding adalah mekanisme di mana kelas anak mendefinisikan ulang metode di kelas induknya.

Contoh Penerapan :

### - Overriding

```
lib > inheritance_overriding.dart > main
Run | Debug | Profile | Qodo Gen: Options | Test this function
1 void main() {
2   Mahasiswa s = new Mahasiswa();
3   s.w1(5);
4 }
5
Qodo Gen: Options | Test this class
6 class Wali {
7   Qodo Gen: Options | Test this method
8   void w1(int x) {
9     print("nilai dari x : ${x}");
10  }
11 }
12
Qodo Gen: Options | Test this class
13 class Mahasiswa extends Wali {
14   @override
15   Qodo Gen: Options | Test this method
16   void w1(int y) {
17     print("nilai dari y : ${y}");
18 }
19 }

PROBLEMS 76 OUTPUT DEBUG CONSOLE TERMINAL PORTS
Selamat Belajar
PS C:\Users\Asus\pertemuan5> dart lib/inheritance_overriding.dart
nilai dari y : 5
PS C:\Users\Asus\pertemuan5>
```

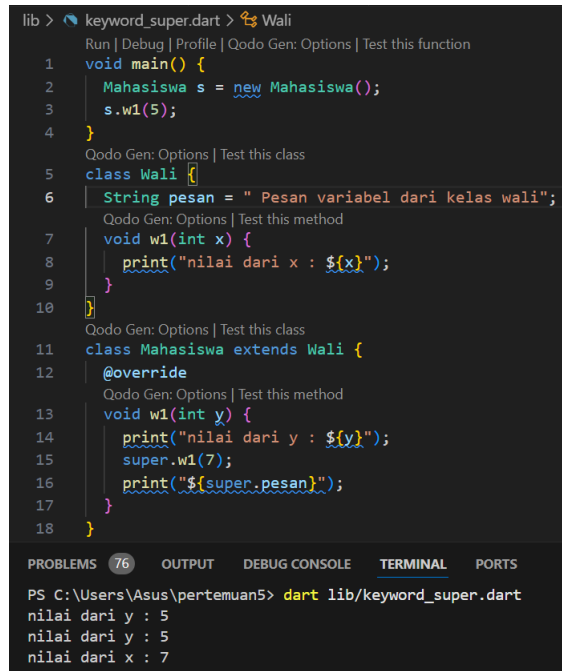
### - Statickey

```
lib > statickey.dart > ...
Qodo Gen: Options | Test this class
1 class StaticVal {
2   static int angka = 0;
3   Qodo Gen: Options | Test this method
4   static view() {
5     print("nilai dari angka adalah ${StaticVal.angka}");
6   }
7 }
8
Run | Debug | Profile | Qodo Gen: Options | Test this function
9 void main() {
10   StaticVal.angka = 5;
11   StaticVal.view();
12 }

PROBLEMS 76 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Asus\pertemuan5> dart lib/statickey.dart
nilai dari angka adalah 5
```

Kata kunci 'static' dapat diterapkan ke anggota data kelas, yaitu bidang dan metode. Variabel statis mempertahankan nilainya sampai program selesai dieksekusi. Anggota statis direferensikan dengan nama kelas.

- Superkey



```
lib > keyword_super.dart > Wali
Run | Debug | Profile | Qodo Gen: Options | Test this function
1 void main() {
2   Mahasiswa s = new Mahasiswa();
3   s.w1(5);
4 }
Qodo Gen: Options | Test this class
5 class Wali {
6   String pesan = " Pesan variabel dari kelas wali";
7   void w1(int x) {
8     print("nilai dari x : ${x}");
9   }
10 }
Qodo Gen: Options | Test this class
11 class Mahasiswa extends Wali {
12   @override
13   void w1(int y) {
14     print("nilai dari y : ${y}");
15     super.w1(7);
16     print("${super.pesan}");
17   }
18 }
PROBLEMS 76 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Asus\pertemuan5> dart lib/keyword_super.dart
nilai dari y : 5
nilai dari y : 5
nilai dari x : 7
```

Super digunakan untuk merujuk ke induk langsung dari suatu kelas. Kata kunci dapat digunakan untuk merujuk ke versi kelas super dari variabel, properti, atau metode.

## PERTEMUAN 7

### Row and Coloumn

#### Contoh Penerapan :

##### - Pepak Jawa

Program ini merupakan kamus sederhana untuk menerjemahkan kata dari Bahasa Indonesia ke Bahasa Jawa. Menggunakan konsep Map/Dictionary untuk menyimpan pasangan kata Indonesia-Jawa.

```
lib > pepak_jawa.dart > ...
1 import 'dart:io';
2
3 void main() {
4   Map<String, Object> indoJawa = {
5     "makan": "mangan",
6     "pulang": "mulih",
7     "aku": "kulo",
8     "sayang": "tresno",
9     "kamu": "kowe",
10   };
11   print("\n kamus jawa sederhana, masukkan salah satu kata dibawah: \n");
12   indoJawa.forEach((key, value) {
13     print(key);
14   });
15   stdout.write("\n ketikkan disini? ");
16   String? indo = stdin.readLineSync();
17   print("\n indonesia => basa jawa \n");
18   print("\n$indo => ${indoJawa[indo]}\n");
19 }

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

kamus jawa sederhana, masukkan salah satu kata dibawah:

makan  
pulang  
aku  
sayang  
kamu

ketikkan disini? []

##### - Cek Bilangan Prima

```
lib > bilangan_prima.dart > ...
1 import 'dart:io';
2
3 void cekPrima(int angkaP) {
4   List<int> a = [];
5   for (var i = 1; i <= angkaP; i++) {
6     if (angkaP % i == 0) a.add(i);
7   }
8   a.length == 2 ? print("Bilangan Prima") : print("Bukan bilangan prima");
9 }
10
11 void main() {
12   stdout.write("masukkan angka:");
13   int angkaPilihan = int.parse(stdin.readLineSync());
14   cekPrima(angkaPilihan);
15 }
16

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\Asus\pertemuan5> dart lib/bilangan\_prima.dart  
masukkan angka:7  
Bilangan Prima

Program meminta user memasukkan sebuah angka

Program akan mengecek apakah angka tersebut merupakan bilangan prima  
Menggunakan konsep list comprehension untuk mencari faktor dari angka tersebut. Bilangan prima adalah bilangan yang hanya memiliki 2 faktor (1 dan bilangan itu sendiri).

## PERTEMUAN 9

### Method Synchronus dan Asynchronus

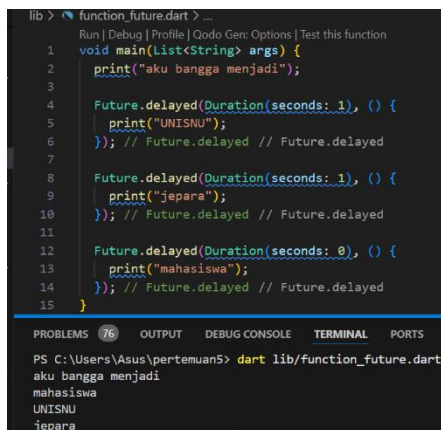
#### Keyword Future

Future merupakan salah satu keyword yang disediakan oleh dart yang seperti namanya yaitu masa depan, jadi keyword ini gunanya untuk mengembalikan masa depan atau mengembalikan nilainya untuk waktu yang akan datang.

Contoh Penerapan :

- Function Future

buatlah output dari coding ini menjadi “aku bangga menjadi mahasiswa UNISNU Jepara” , tidak perlu menggunakan `async` `await`.



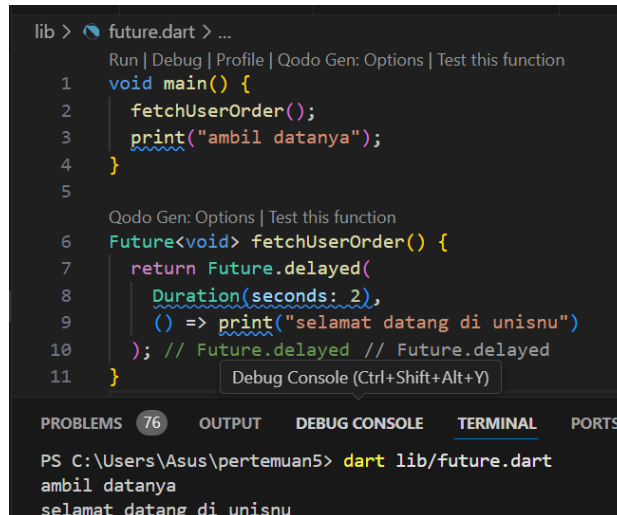
```
lib > function_future.dart > ...
Run | Debug | Profile | Qodo Gen: Options | Test this function
1 void main(List<String> args) {
2   print("aku bangga menjadi");
3
4   Future.delayed(Duration(seconds: 1), () {
5     print("UNISNU");
6   }); // Future.delayed // Future.delayed
7
8   Future.delayed(Duration(seconds: 1), () {
9     print("jepara");
10  }); // Future.delayed // Future.delayed
11
12  Future.delayed(Duration(seconds: 0), () {
13    print("mahasiswa");
14  }); // Future.delayed // Future.delayed
15 }

PROBLEMS 76 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Asus\pertemuan5> dart lib/function_future.dart
aku bangga menjadi
mahasiswa
UNISNU
jepara
```

Penerapan Fungsi Future delayed digunakan untuk menunda eksekusi cetakan (output) sesuai durasi yang ditentukan. Program ini mencetak pesan secara *asynchronous*: "aku bangga menjadi" dicetak lebih dulu karena tidak ada penundaan, lalu "mahasiswa" (dengan durasi 0 detik), dan terakhir "UNISNU" serta "jepara" yang ditunda masing-masing selama 1 detik.

- Future.dart

Future merupakan salah satu keyword yang disediakan oleh dart yang seperti namanya yaitu masa depan, jadi keyword ini gunanya untuk mengembalikan masa depan atau mengembalikan nilainya untuk waktu yang akan datang.



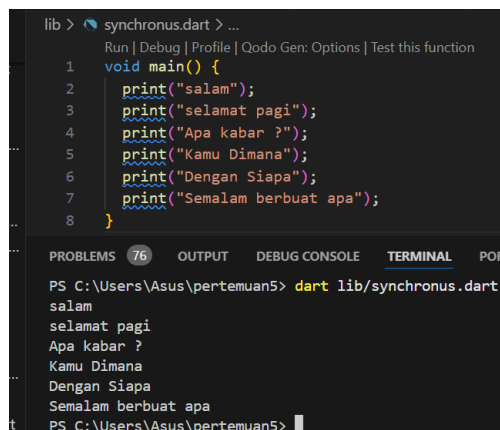
```
lib > future.dart > ...
Run | Debug | Profile | Qodo Gen: Options | Test this function
1 void main() {
2   fetchUserOrder();
3   print("ambil datanya");
4 }
5
Qodo Gen: Options | Test this function
6 Future<void> fetchUserOrder() {
7   return Future.delayed(
8     Duration(seconds: 2),
9     () => print("selamat datang di unisnu")
10  ); // Future.delayed // Future.delayed
11 }
Debug Console (Ctrl+Shift+Alt+Y)

PROBLEMS 76 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Asus\pertemuan5> dart lib/future.dart
ambil datanya
selamat datang di unisnu
```

- Synchronus.dart

Synchronus merupakan default dalam pemrosesan kode, artinya setiap kode yang ditulis akan dieksekusi baris perbaris. Bisa diartikan baris kode tidak akan dieksekusi jika baris kode sebelumnya belum dieksekusi.

Contoh Penerapan :



```
lib > synchronus.dart > ...
Run | Debug | Profile | Qodo Gen: Options | Test this function
1 void main() {
2   print("salam");
3   print("selamat pagi");
4   print("Apa kabar ?");
5   print("Kamu Dimana");
6   print("Dengan Siapa");
7   print("Semalam berbuat apa");
8 }

PROBLEMS 76 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Asus\pertemuan5> dart lib/synchronus.dart
salam
selamat pagi
Apa kabar ?
Kamu Dimana
Dengan Siapa
Semalam berbuat apa
PS C:\Users\Asus\pertemuan5>
```



## - Asynchronous

Asynchronous merupakan proses kode yang eksekusinya tidak harus menunggu/ sesuai urutan baris kode tetapi berdasarkan waktu proses.

Asynchronous berguna saat digunakan untuk :

1. Mengambil data melalui jaringan internet
2. Menulis atau mengambil data dari database
3. Membaca data dari file

Untuk menggunakan asynchronous di Dart jangan lupa gunakan keyword `async` dan `await`.

## - Contoh Penerapan



```
lib > asynchronous.dart > Mahasiswa > dataMahasiswa
Run | Debug | Profile | Qodo Gen: Options | Test this function
1 void main() {
2   var data = Mahasiswa();
3
4   print('hallo');
5   print('selamat datang');
6   print('tuan${data.nama}');
7   print('apa kabar');
8 }
9
Qodo Gen: Options | Test this class
10 class Mahasiswa {
11   var nama = "tanpa nama";
12
13   Qodo Gen: Options | Test this method
14   Future<void> dataMahasiswa() async {
15     await Future.delayed(Duration(seconds: 4));
16     nama = "Rasya";
17   }
18 }

PROBLEMS 76 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Asus\pertemuan5> dart lib/asynchronous.dart
hallo
selamat datang
tuan tanpa nama
apa kabar
```

## PERTEMUAN 10

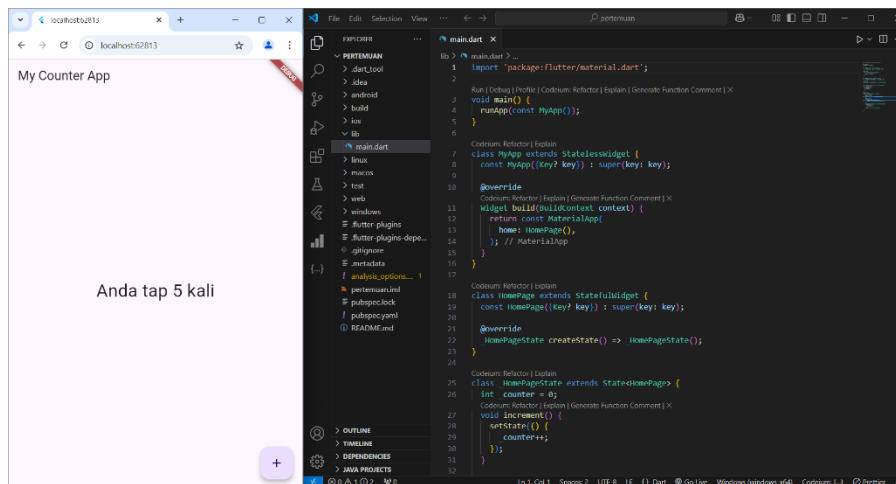
### Flutter

#### Intro Flutter

Flutter adalah software development kit (SDK) buatan google yang berfungsi untuk membuat aplikasi mobile menggunakan bahasa pemrograman Dart, baik untuk Android maupun iOS. Dengan Flutter, aplikasi Android dan iOS dapat dibuat menggunakan basis kode dan bahasa pemrograman yang sama, yaitu Dart, bahasa pemrograman yang juga diproduksi oleh Google pada tahun 2011. Sebelumnya, aplikasi murni (native) untuk Android perlu dibuat menggunakan bahasa Java atau Kotlin, sedangkan aplikasi iOS perlu dibuat menggunakan bahasa pemrograman Objective-C atau Swift. Flutter ditunjukan untuk mempermudah dan mempercepat proses pengembangan aplikasi mobile yang dapat berjalan di atas Android dan iOS, tanpa harus mempelajari dua bahasa pemrograman secara terpisah.

#### Membuat Aplikasi Counter dari NOL Flutter

#### Contoh Penerapan :



sekarang setiap anda menekan FloatingActionButton + angka akan bertambah dan UI juga berubah.

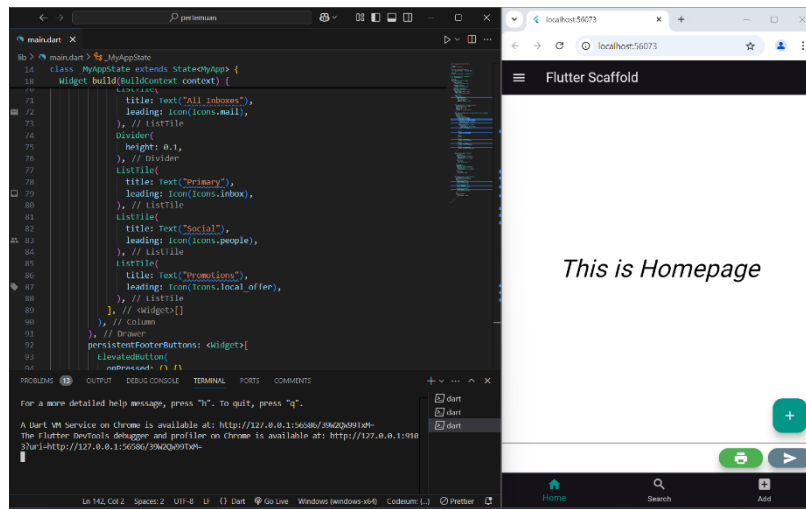
## PERTEMUAN 11

### Flutter

#### Penerapan Scaffold Widget

Widget Scaffold menyediakan kerangka kerja yang mengimplementasikan dasar dari material design yaitu struktur layout visual dari aplikasi flutter. Ia menyediakan API untuk menampilkan drawers, snack bars dan bottom sheets.

Contoh Penerapan :



## PERTEMUAN 12

### Flutter

#### Flutter Navigation

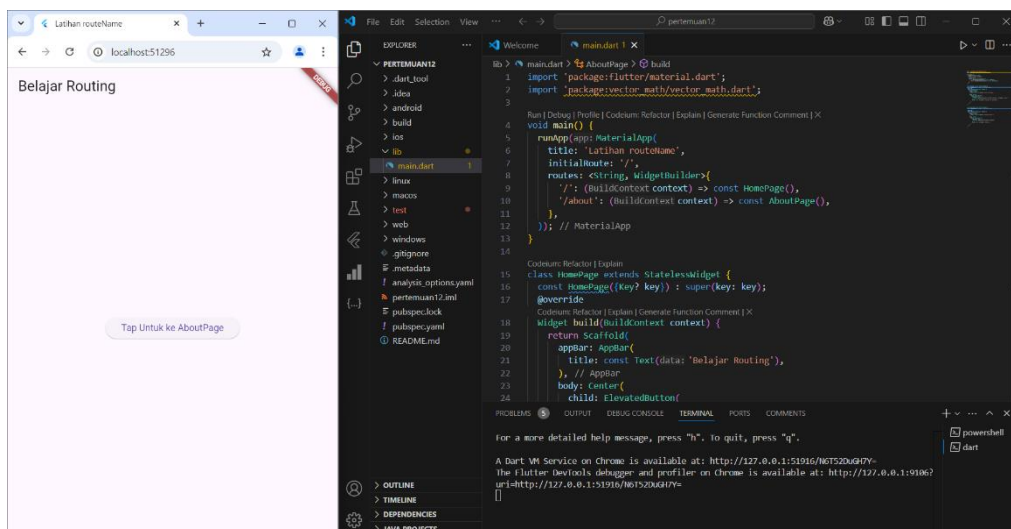
Dalam sebuah aplikasi mobile biasanya memiliki full-screen elemen yang disebut “screen” atau “page”. Di Flutter, elemen ini disebut route dan dikelola oleh widget Navigator. Widget navigator berfungsi untuk menampilkan konten ke halaman atau layar baru. Jika pada native Android, Navigator route dinamakan dengan activity dan di iOS sebagai viewController.

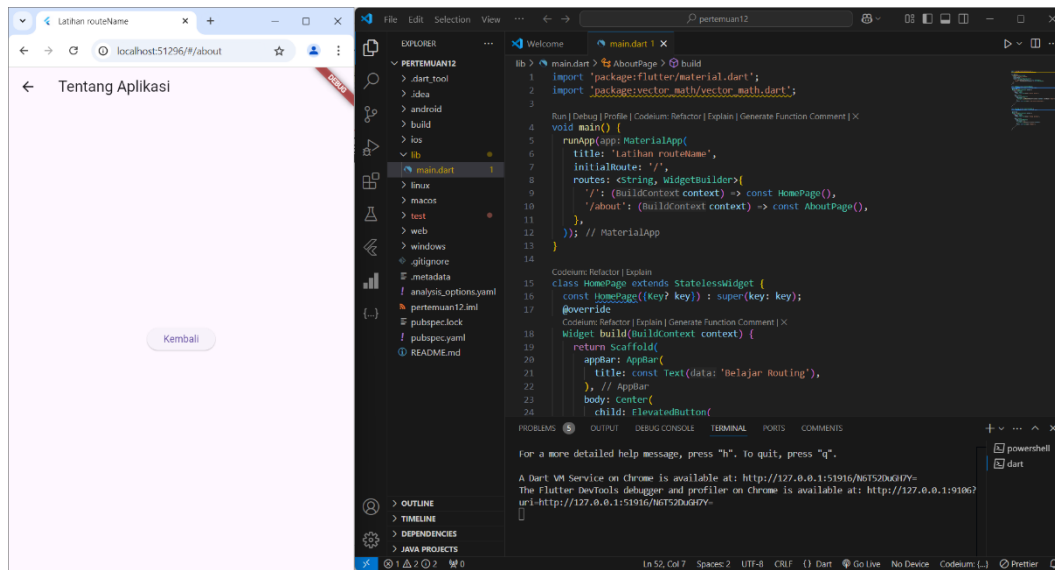
Widget Navigator bekerja seperti tumpukan layar (stack), ia menggunakan prinsip LIFO (Last-In, First-Out). Ada dua method yang dapat digunakan pada Navigator widget yaitu :

1. `Navigator.push ()`: Metode push digunakan untuk menambahkan rute lain ke atas tumpukan screen (stack) saat ini. Halaman baru ditampilkan di atas halaman sebelumnya.
2. `Navigator.pop ()`: Metode pop menghapus rute paling atas dari tumpukan. Ini menampilkan halaman sebelumnya kepada pengguna.

Contoh Penerapan :

- Membuat Simple Routing Flutter





Membuat dua stateless widget dimana halaman awal menggunakan HomePage dan berpindah ke AboutPage saat tombol di tap.

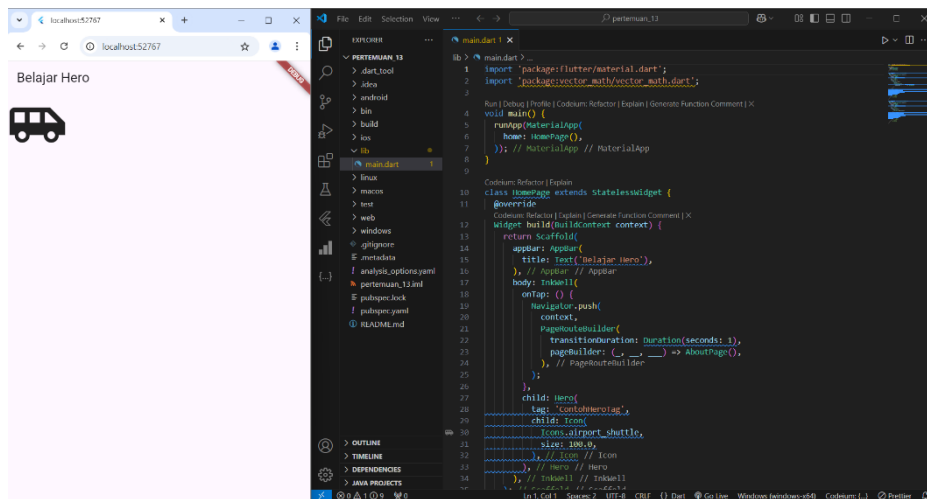
## Pertemuan 13

### Flutter

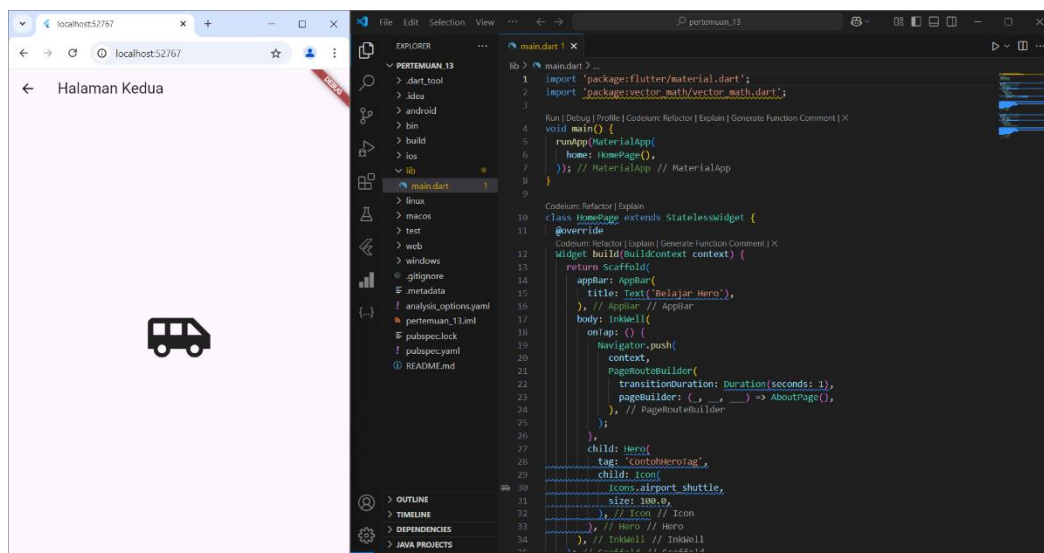
#### Hero Animation

Lanjutan Materi Flutter Navigation pekan lalu, Hero Animasi merupakan sebuah element transisi yang bergerak melayang saat berpindah dari satu layar ke layar lainnya.

Contoh Penerapan:



Saat icon mobil diklik dia akan bergerak dan tampil seperti gambar dibawah ini



Hero Animation bisa dikatakan merupakan salah satu animasi paling mudah dan sederhana di flutter. Caranya yaitu hanya dengan menggunakan class **Hero** pada kedua item (awal dan akhir) dan dihubungkan dengan properti **tag** yang memilik

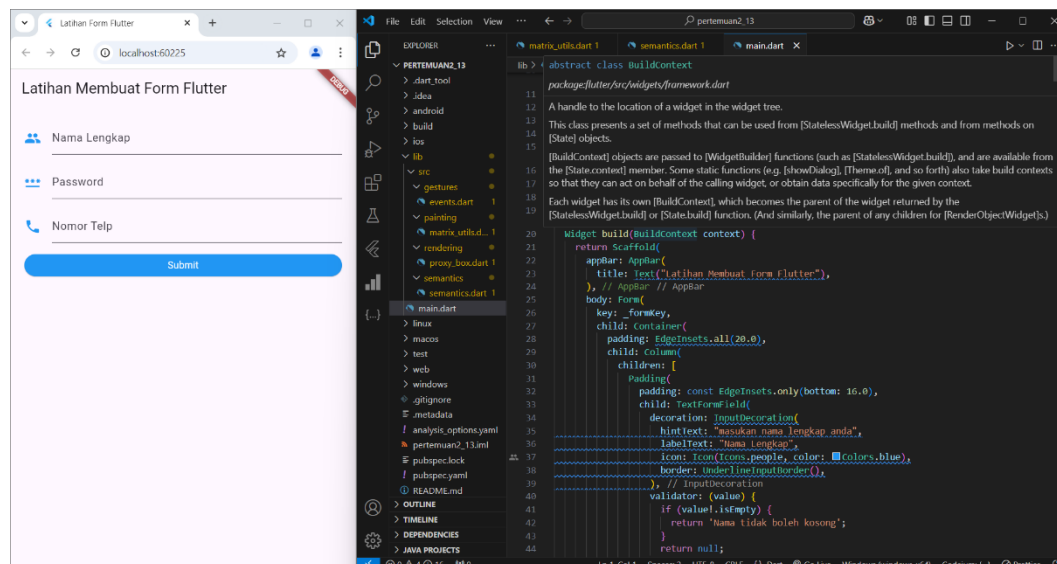
nilai yang sama. Kedua pasangan masing element harus memiliki value tag yang sama dan bersifat unik. Sebagai contoh kita akan membuat transisi icon mobil dimana posisi awal mobil berada di pojok kiri atas dan akan bergerak ketengah saat berpindah ke layar selanjutnya.

## Form di Flutter

Form merupakan hal yang umum ditemukan dan penting dalam sebuah aplikasi mobile. Penggunaan form juga sangat beragam, dari mulai untuk form login, register, kolom komentar, halaman order, dan banyak lagi. Karena pentingnya memahami penggunaan form pada sebuah aplikasi mobile, untuk itu dalam kesempatan ini kita akan membahas cara membuat form di Flutter menggunakan Form widget beserta komponen dan widget di dalamnya.

Form widget sendiri berfungsi untuk mempermudah dalam proses pembuatan dan memberi keamanan lebih pada aplikasi flutter seperti validasi, dan aksi lainnya yang umum terdapat pada sebuah form

### Contoh Penerapan :



## PERTEMUAN 14

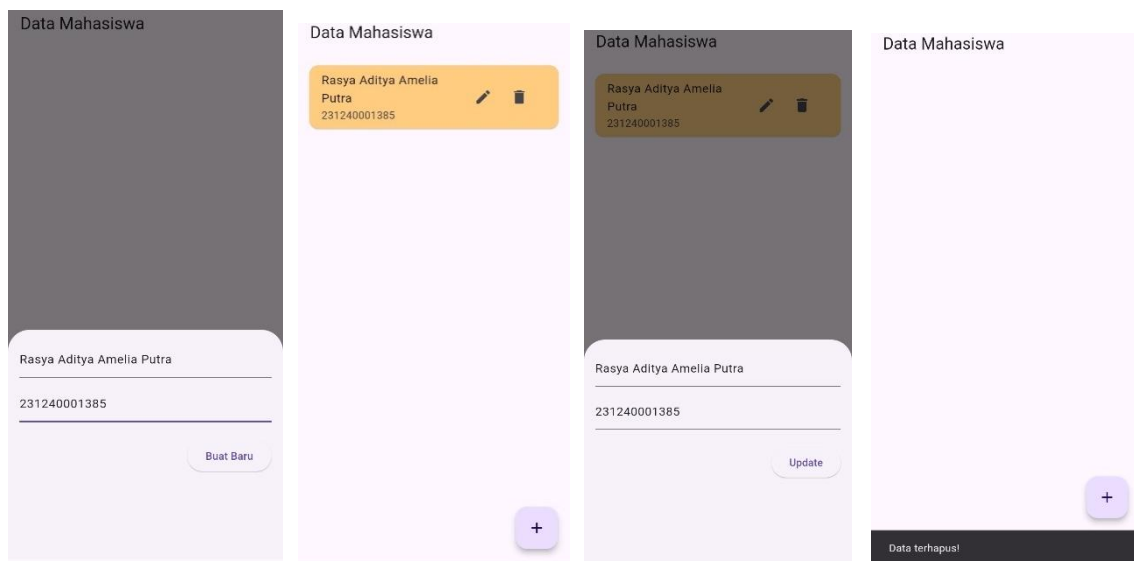
### Sqllite

#### Sqllite

Sqllite adalah paket perangkat lunak domain publik yang menyediakan sistem manajemen Database relasional, atau RDBMS. Database relasional digunakan untuk menyimpan catatan yang ditentukan pengguna dalam tabel besar. Selain penyimpanan dan manajemen data, mesin database dapat memproses perintah kueri kompleks yang menggabungkan data dari beberapa tabel untuk menghasilkan laporan dan ringkasan data.

Produk RDBMS populer lainnya termasuk Oracle Database, IBM DB2, dan Microsoft SQL Server di sisi komersial, dengan MySQL dan PostgreSQL menjadi produk open source yang populer.

Contoh Penerapan:



Disini Kita dapat menambah , edit dan hapus data.



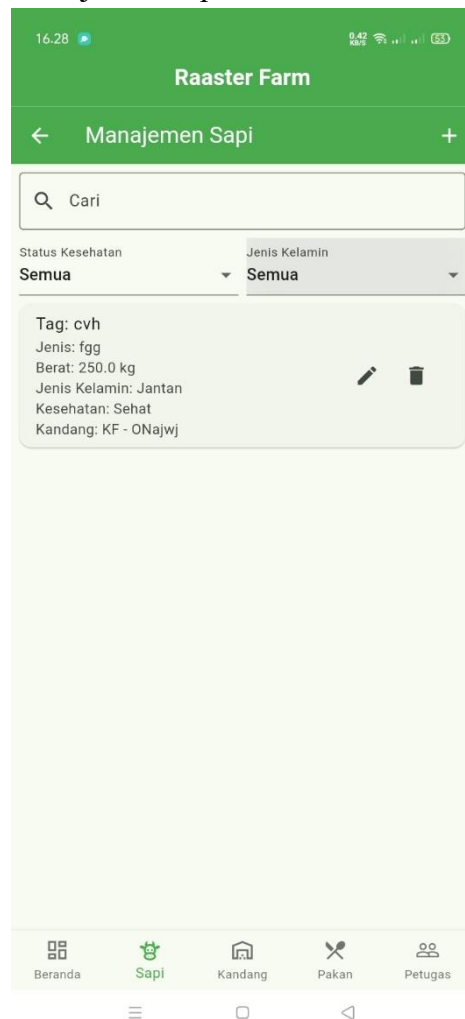
## UJIAN AKHIR SEMESTER

Membuat Aplikasi CRUD Menggunakan Flutter Ide proyek akhir saya adalah Ide proyek akhir saya adalah membuat aplikasi Sistem Manajemen Peternakan Sapi. Database yang saya gunakan adalah SQLite. Dengan adanya Aplikasi Sistem Manajemen Peternakan Sapi, aplikasi ini berguna untuk menyimpan data sapi beserta jenisnya, agar peternak dapat dengan mudah memantau informasi hewan ternak. Selain itu, aplikasi ini membantu mencatat riwayat kesehatan sapi, seperti vaksinasi dan pemberian pakan, sehingga mempermudah pengelolaan peternakan secara digital. Berikut Demo Aplikasinya Tampilan Awal (Dashboard)

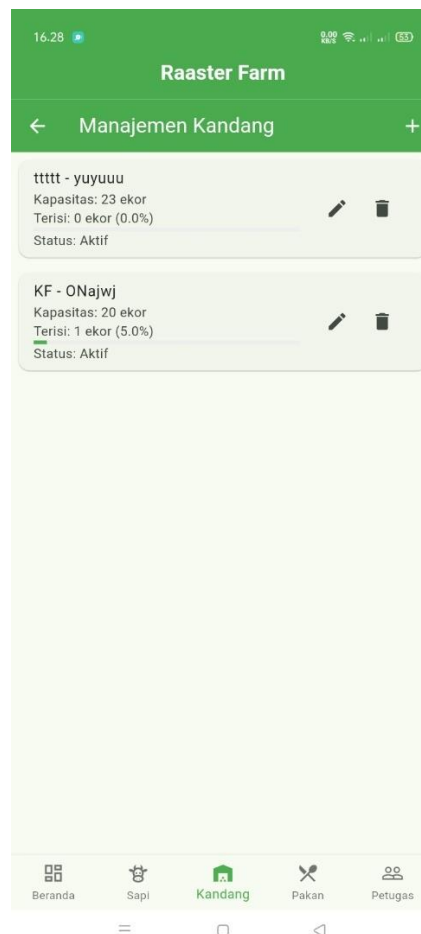


Berikut adalah beberapa Fitur yang ada di aplikasi Raaster Farm :

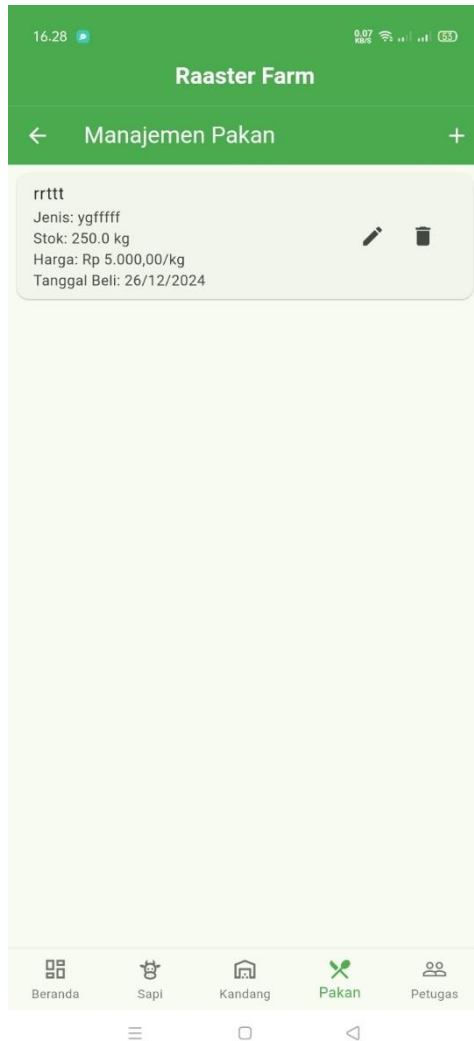
#### -Manajemen Sapi



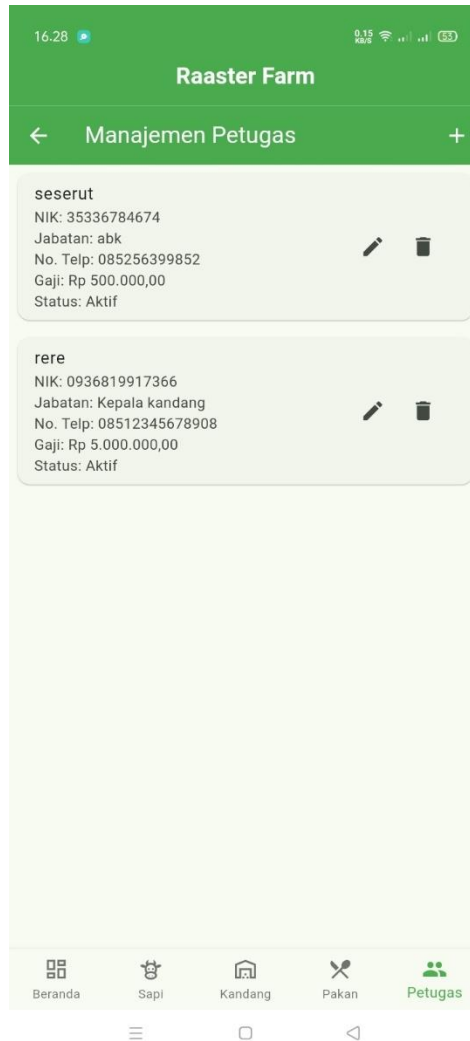
#### -Manajemen Kandang



## -Manajemen Pakan



## -Manajemen Petugas



## DAFTAR PUSTAKA

- Handoyo, Darmawan Erico. 2020. *Pemrograman Berorientasi Objek menggunakan Bahasa Pemrograman DART*. Jakarta: Penerbit Informatika..
- Raharjo, Budi. 2019. *Pemrograman Berorientasi Objek dengan Flutter*. Bandung: Informatika Bandung.
- Wahyono, Hadi. 2020. *Flutter untuk Pemula: Dari Dasar Hingga Membuat Aplikasi*. Jakarta: PT Gramedia Pustaka Utama.