**Dart Fundamental**

Kita telah belajar bagaimana membuat aplikasi sederhana “Hello, World!” menggunakan Dart. Pada modul ini kita akan mempelajari konsep-konsep dasar pada Dart. Beberapa topik yang akan kita pelajari antara lain variabel, tipe data, *operator*, hingga *functions*.

## Comments

Sebelum mulai mengembangkan aplikasi yang lebih kompleks, ada satu hal penting lagi yang perlu kita tahu, yaitu instruksi kepada komputer untuk mengabaikan bagian dari suatu program. Kode yang ditulis dalam suatu program tetapi tidak dijalankan oleh komputer disebut “comments.”

Sebuah komentar akan dilewatkan ketika proses kompilasi, sehingga tidak akan memengaruhi alur program yang kita tulis. Komentar bisa digunakan sebagai dokumentasi yang menjelaskan kode yang kita tulis.

Terdapat tiga jenis komentar yang bisa digunakan pada Dart. Pertama adalah ***single-line comment*** atau komentar satu baris yang diawali dengan tanda // dan berakhir pada akhir baris tersebut.

1. // Single line comment

Selanjutnya ada ***multi-line comment*** di mana kita bisa menuliskan beberapa baris komentar. Komentar ini dimulai dari tanda /\* dan diakhiri dengan \*/.

1. /\*
2. multi
3. line
4. comment
5. \*/

Kemudian terakhir adalah ***documentation comment***. Komentar ini adalah single-line comment atau multi-line comment yang diawali dengan /// atau /\*\*.

Di dalam documentation comment, kompiler Dart akan mengabaikan semua teks kecuali yang tertutup dalam kurung siku ([]). Di dalam kurung siku kita dapat memasukkan referensi dari nama kelas, variabel, atau fungsi. Berikut ini adalah contoh penggunaan komentar.

## Variables

Ketika menulis sebuah program, kita memberi tahu komputer cara memproses informasi seperti mencetak teks ke layar atau melakukan operasi perhitungan. Untuk lebih memudahkan penggunaan dan pemanggilan data ini maka kita bisa memanfaatkan variabel. Variabel bisa dibayangkan sebagai sebuah kotak atau wadah yang menyimpan nilai. Di dalam komputer variabel ini disimpan di dalam memori komputer. Setiap variabel memiliki nama yang dapat kita panggil dan gunakan.

Pada Dart kita mendefinisikan sebuah variabel dengan keyword var.

1. var greetings = 'Hello Dart!';

Perhatikan tanda sama dengan (=) di atas. Simbol tersebut dikenal dengan ***assignment operator***. Kode di atas berarti kita memasukkan nilai ‘Hello Dart!’ ke dalam sebuah kotak atau variabel yang bernama greetings. Proses assignment nilai ke variabel ini disebut inisialisasi.

Selanjutnya mari coba tampilkan nilai variabel ini ke konsol. Ubah kode fungsi main Anda menjadi seperti berikut.

void **main**() {

  var myName = 'Fikri';

  var myAge;

  myAge = 20;

**print**('My Name Is : $myName My Age Is :$myAge');

**Data Types**

Pada materi sebelumnya kita telah mempelajari tentang variabel yang dapat menyimpan nilai. Jadi bagaimana komputer membedakan antara variabel yang bernilai angka atau teks? Dan kenapa penting untuk bisa membedakannya?

Dart memiliki banyak tipe data yang mewakili jenis data yang dapat kita gunakan dan bagaimana data tersebut dioperasikan. Dengan tipe data, komputer dapat menghindari operasi yang tidak mungkin serta bisa menghasilkan *bug*, misalnya seperti perkalian karakter alfabet atau mengubah angka menjadi kapital.

Dart adalah bahasa yang mendukung *type inference*. Ketika Anda mendeklarasikan variabel dengan var, Dart akan secara otomatis menentukan tipe datanya. Misalnya :

1. var greetings = 'Hello Dart!';  // String
2. var myAge = 20;               // integers

Komputer akan tahu bahwa variabel greetings memiliki nilai berupa String atau teks dan variabel myAge bernilai angka atau integers meskipun kita tidak mendefinisikannya secara eksplisit.

Anda tetap bisa mendeklarasikan tipe data variabel secara eksplisit untuk menghindari kebingungan dan memudahkan proses *debugging*.

1. String greetings = 'Hello Dart!';
2. int myAge = 20;

Beberapa tipe data yang didukung oleh Dart antara lain:

| **Tipe** | **Deskripsi** | **Contoh** |
| --- | --- | --- |
| int | Integer (bilangan bulat) | 5, -7, 0 |
| double | Bilangan desimal | 3.14, 18.0, -12.12 |
| num | Bilangan bulat dan bilangan desimal | 5, 3.14, -99.00 |
| bool | Boolean | true, false |
| String | Teks yang terdiri dari 0 atau beberapa karakter | ‘Dicoding’, ‘Y’, ‘’ |
| List | Daftar nilai | [1, 2, 3], [‘a’, ‘b’, ‘c’] |
| Map | Pasangan key-value | {“x”: 4, “y”: 10} |
| dynamic | Tipe apa pun |  |

Dart mendukung *type inference*, menariknya ketika kita mendeklarasikan variabel tanpa melakukan inisialisasi, variabel akan memiliki tipe ***dynamic***. Karena sebuah variabel bernilai *dynamic* bisa berisi tipe apa pun, maka tidak ada masalah jika kita mengubah nilai di dalamnya.

void **main**() {

  var x; // dynamic

  x = 7;

  x = 'Dart is great';

**print**(x);

}

Kode di atas tetap bisa berjalan dan menampilkan pesan ‘Dart is great’ tanpa ada masalah. Berbeda jika kita menginisialisasi nilai variabel x secara langsung. Akibatnya, editor akan menampilkan eror karena terjadi perubahan tipe data.

### **Menerima input pengguna**

Selain menampilkan pesan ke konsol, kita juga dapat menerima input pengguna untuk selanjutnya diproses dan ditampilkan. Ini memungkinkan kita dapat membuat aplikasi yang interaktif dengan pengguna.

Untuk bisa menerima input, kita akan menggunakan statement stdin.readLineSync(). Fungsi ini merupakan bagian dari library dart:io, sehingga kita perlu mengimpor library tersebut.

1. import 'dart:io';

Kali ini kita akan membuat aplikasi sederhana yang menerima input nama dan usia dari pengguna lalu menampilkan pesan ke konsol.

### **Kode solusi**

Pada persoalan di atas kita perlu menampilkan output, menerima beberapa input, dan menampilkan output lagi sesuai input yang diberikan. Kode yang perlu Anda tulis kurang lebih adalah seperti berikut:

1. import 'dart:io';
3. void main() {
4. stdout.write('Nama Anda : ');
5. String name = stdin.readLineSync()!;
6. stdout.write('Usia Anda : ');
7. int age = int.parse(stdin.readLineSync()!);
8. print('Halo $name, usia Anda $age tahun');
9. }

Jika kode Anda berbeda karena menggunakan print() maka tidak masalah. Statement print() dan stdout.write() memiliki fungsi yang sama yaitu untuk menampilkan suatu objek ke konsol. Yang membedakan adalah print() akan mencetak baris baru setelah menampilkan sesuatu sehingga selanjutnya Anda perlu memasukkan input pada baris baru. Sementara stdout.write() hanya menampilkan objeknya saja dan ketika ada input atau output baru lagi masih akan ditampilkan di baris yang sama.

Selain itu, kita menambahkan tanda ! setelah stdin.readLineSync() untuk menandai bahwa input tidak akan mengembalikan nilai null. Penjelasan terkait nilai null akan kita bahas lebih detail nanti pada materi null safety.

Kode baru lain adalah int.parse(). Kita menggunakan kode ini untuk mengkonversi tipe data String menjadi int. Input yang diambil dari stdin.readLineSync() akan memiliki tipe data berupa String. Sehingga ketika ingin menyimpan dan mengoperasikan input dalam tipe data lain kita perlu melakukan konversi terlebih dahulu.

## Numbers

Tipe data angka pada Dart dapat disimpan ke dalam dua jenis: ***int*** dan ***double***.

Integers adalah nilai bilangan bulat yang tidak lebih besar dari 64 bit tergantung platform yang digunakan. Untuk Dart VM variabel integer dapat menyimpan nilai mulai dari -2^63 hingga 2^63 - 1, sementara jika dikompilasi ke JavaScript integer memiliki nilai dari -2^53 sampai 2^53 - 1.

Integer atau bilangan bulat adalah bilangan yang tidak memiliki titik desimal. Contohnya seperti berikut:

1. var number = 2020;
2. var hex = 0xDEADBEEF;

Jika sebuah bilangan adalah bilangan desimal, maka bilangan tersebut termasuk tipe data double. Double juga memiliki ukuran sebesar 64 bit. Berikut ini adalah contoh variabel double:

1. var decimal = 1.2;
2. var pi = 3.14;

Sejak versi Dart 2.1, kita bisa menuliskan tipe data double tanpa menuliskan angka di belakang koma secara literal. Sebelum versi tersebut, kita harus menuliskan bilangan desimal secara literal.

1. double withoutDecimal = 7; // Sama dengan double withoutDecimal = 7.0

Int dan double adalah subtipe dari tipe data num. Ketiga tipe data ini dapat kita gunakan untuk melakukan perhitungan dasar seperti penjumlahan, perkalian, hingga menggunakan fungsi seperti abs(), ceil(), floor(), dan banyak fungsi lainnya. Jika Anda membutuhkan operasi perhitungan namun tidak tersedia pada tiga tipe data ini maka Anda bisa memanfaatkan library dart:math.

Pada materi sebelumnya kita memiliki kebutuhan untuk mengonversi tipe String menjadi int. Konversi tipe data ini adalah hal yang penting dan kita akan banyak membutuhkannya. Selain konversi String menjadi int, kita juga bisa melakukan konversi double menjadi String atau sebaliknya.

void **main**() {

  // String -> int

  var eleven = int.**parse**('11');

**print**(eleven.runtimeType);

  // String -> double

  var elevenPointTwo = double.**parse**('11.2');

**print**(elevenPointTwo.runtimeType);

  // int -> String

  var elevenAsString = 11.**toString**();

**print**(elevenAsString.runtimeType);

  // double -> String

  var piAsString = 3.14159.**toStringAsFixed**(2); // String piAsString = '3.14'

**print**(piAsString.runtimeType);

}

## Strings

Kita telah banyak menggunakan String pada materi-materi sebelumnya dan seperti kita tahu, String digunakan untuk menyimpan data dalam bentuk teks. String pada Dart dapat didefinisikan dengan menggunakan tanda petik satu maupun petik dua.

1. String singleQuote = 'Ini adalah String';
2. String doubleQuote = "Ini juga String";

Anda dapat menggunakan tanda petik secara bergantian, khususnya jika Anda memiliki teks yang mengandung tanda petik.

Lalu bagaimana jika teks kita memiliki kedua tanda petik ini?

1. print('"I think it's great!" I answered confidently');

Tentunya kode di atas akan menghasilkan eror. Solusinya, gunakanlah backslash (\) untuk mengurangi ambiguitas dalam tanda petik. Mekanisme ini juga dikenal dengan nama ***escape string***. Sehingga kode di atas akan menjadi seperti berikut:

1. void main() {
2. print('"I think **it\'s** great!" I answered confidently');
3. }

Backslash sebelum tanda petik ini akan mengatakan kepada Dart bahwa itu hanyalah tanda petik dan tidak boleh ditafsirkan sebagai pembatas string. Selain tanda petik, backslash juga berguna untuk mengabaikan simbol lain yang menimbulkan ambigu di dalam string, contohnya seperti backslash itu sendiri.

1. void main() {
2. print("Windows path: **C:\\Program Files\\Dart**");
3. }

Fitur lain dari String adalah ***String interpolation*** di mana kita bisa memasukkan nilai dari variabel atau expression ke dalam string. Interpolation ini bisa kita terapkan dengan simbol $.

Jika Anda perlu menampilkan harga dalam dollar, maka apa yang akan Anda lakukan? Yup, backslash akan memberitahu Dart mana tanda $ sebagai interpolation atau sebagai mata uang. Selain itu Anda juga menggunakan cara seperti berikut:

1. void main() {
2. print(**r'Dia baru saja membeli komputer seharga $1,000.00'**);
3. }

Huruf ‘***r***’ sebelum String akan memberitahu Dart untuk menganggap String sebagai raw, yang berarti akan mengabaikan interpolation.

Selain itu, kita juga bisa menambahkan sebuah Unicode ke dalam String. Pada Dart Unicode ini dikenal dengan ***runes***. Unicode mendefinisikan nilai numerik unik untuk setiap huruf, angka, dan simbol yang digunakan dalam semua sistem penulisan dunia. Cara umum untuk mengekspresikan unicode adalah \uXXXX, di mana ***XXXX*** adalah nilai heksadesimal 4 digit. Misalnya karakter hati (♥) adalah \u2665.

## Booleans

Setelah angka dan teks, ada satu tipe data utama lagi yang penting untuk dipelajari, yaitu ***boolean***. Nama boolean ini diambil dari nama seorang matematikawan asal Inggris yang bernama George Boole. Beliau dikenal karena penciptaan aljabar boolean, yakni cabang aljabar di mana nilai variabel selalu benar atau salah.

Boolean pada Dart dideklarasikan dengan kata kunci bool. Sesuai dengan penjelasan di atas, variabel boolean hanya bisa menyimpan dua nilai, yaitu true dan false.

1. bool alwaysTrue = true;
2. var alwaysFalse = false;
3. var notTrue = !true;
4. bool notFalse = !false;

Tanda ! di atas disebut dengan operator “***not***” atau “***bang***”. Operator ini berfungsi untuk menegasikan nilai boolean, sederhananya membalik nilai boolean. Misalnya !true sama saja bernilai false.

Kita akan banyak menggunakan boolean dalam pengembangan aplikasi untuk operasi perbandingan dan juga pengondisian aplikasi.

void **main**() {

  if (true) {

**print**("It's true");

  } else {

**print**("It's False");

  }

}

## Operators

Istilah operator dipinjam dari matematika dengan pengertian yang sedikit berbeda. Pada Dart operator menginstruksikan komputer untuk melakukan operasi.

Sebenarnya kita telah menerapkan beberapa contoh operator pada materi sebelumnya. Sebagai contoh operator assignment (=) yang kita gunakan untuk inisialisasi nilai variabel.

1. var firstNumber = 4;

Pada kode di atas kita menginstruksikan komputer untuk memasukkan nilai 4 ke dalam variabel firstNumber.

### **Operator aritmatika**

Contoh operator lain yang telah Anda lihat adalah operator aritmatika yang digunakan untuk operasi seperti penjumlahan atau perkalian. Lihatlah contoh kode berikut:

1. void main() {
2. var firstNumber = 4;
3. var secondNumber = 13;
4. var sum = **firstNumber + secondNumber**;
5. print(sum);
6. }

Dart mendukung operator aritmatika umum sebagai berikut:

| **Operator** | **Deskripsi** |
| --- | --- |
| + | Penjumlahan |
| - | Pengurangan |
| \* | Perkalian |
| / | Pembagian |
| ~/ | Pembagian, mengembalikan nilai int |
| % | Modulo atau sisa hasil bagi |

void **main**() {

**print**(5 + 2); // int add = 7

**print**(5 - 2); // int subtract = 3

**print**(5 \* 2); // int multiply = 10

**print**(5 / 2); // double divide = 2.5

**print**(5 ~/ 2); // int intDivide = 2

**print**(5 % 2); // int modulo = 1

}

Operator aritmatika pada pemrograman memiliki aturan yang sama dengan matematika, di mana perkalian dan pembagian akan didahulukan sebelum penjumlahan atau pengurangan.

1. print(2 + 4 \* 2); // output: 10

Jika Anda ingin melakukan operasi penjumlahan terlebih dahulu, gunakan tanda kurung atau parentheses.

1. print((1 + 3) \* (4 - 2)); // output: 8

Selain itu Dart mendukung increment dan decrement. Contohnya adalah seperti berikut:

1. var a = 0, b = 5;
2. a++;
3. b--;
4. print(a); // output = 1
5. print(b); // output = 4

Expression a++ di atas dapat diartikan dengan a = a + 1. Komputer akan mengambil nilai dari a kemudian menambahkan 1 lalu memasukkannya kembali ke variabel a. Bentuk increment lainnya adalah seperti berikut:

1. var c = 0;
2. c += 5; // c = c + 5 atau c = 0 + 5
3. print(c); // output 5

Operator ini juga bisa digunakan pada operator aritmatika lain seperti perkalian dan pembagian.

1. var d = 2;
2. d \*= 3;   // d = d \* 3 atau d = 2 \* 3
3. print(d); // output = 6

### **Operator perbandingan**

Dart juga mendukung operasi perbandingan untuk membandingkan nilai-nilai yang dijadikan sebagai operands. Berikut ini adalah contoh operator perbandingan pada Dart:

| **Operator** | **Deskripsi** |
| --- | --- |
| == | Sama dengan |
| != | Tidak sama dengan |
| > | Lebih dari |
| < | Kurang dari |
| >= | Lebih dari sama dengan |
| <= | Kurang dari sama dengan |

Operator perbandingan ini akan mengembalikan nilai dalam bentuk boolean.

void **main**() {

  if (2 <= 3) {

**print**('Ya, 2 kurang dari sama dengan 3');

  } else {

**print**("Anda salah");

  }

}

### Operator logika

Kita juga dapat menggabungkan ekspresi boolean atau membaliknya dengan operator logika. Operator ini meliputi:

| **Operator** | **Deskripsi** |
| --- | --- |
| || | OR |
| && | AND |
| ! | NOT |

Kita telah membahas operator NOT atau bang pada materi boolean. Sementara itu operator OR atau AND digunakan untuk menguji logika dari beberapa nilai boolean. Operator AND akan menghasilkan nilai true jika semua operand-nya bernilai true, sedangkan OR jika salah satu saja dari operand bernilai true maka operator akan mengembalikan nilai true. Contohnya seperti kode berikut.

**if** (false || true || false) {

**print**('Ada satu nilai true');

  } **else** {

**print**('Jika semuanya false, maka ini akan tampil');

  }

}

## Functions

Functions pada Dart digunakan untuk menghasilkan output berdasarkan input tertentu yang diberikan, selain itu juga sebagai blok kode atau prosedur yang dapat digunakan kembali. Sadar atau tidak, sebenarnya kita telah mengimplementasikan beberapa functions pada kode kita. Semua program Dart dimulai dari fungsi main(). main() adalah contoh fungsi utama yang selalu kita gunakan. Selain itu, print() juga termasuk fungsi.

1. print('Hello Dart!');

Fungsi print() akan mengambil nilai String atau objek lainnya dan menampilkannya ke konsol. Untuk mencetak sesuatu ke konsol sebenarnya dibutuhkan beberapa instruksi yang lebih low-level, namun kita menjadi sangat terbantu dengan adanya fungsi print() ini dan dapat menggunakannya secara berulang.

Untuk mendeklarasikan fungsi, caranya sama dengan penulisan fungsi main() yaitu dengan menentukan tipe nilai balik atau return value lalu nama fungsi dan parameter inputnya.

1. returnType functionName(type param1, type param2, ...) {
2. return result;
3. }

Setiap fungsi Dart selalu mengembalikan nilai. Namun ada satu tipe data khusus yang bisa kita lihat pada fungsi main yaitu return type ***void***. Keyword void berarti fungsi tersebut tidak menghasilkan output atau nilai kembali. Biasanya fungsi seperti ini digunakan untuk kumpulan instruksi atau prosedur yang berulang dan sering digunakan.

Setelah fungsi dibuat, selanjutnya kita bisa memanggilnya pada fungsi main() atau pada bagian program lain yang Anda inginkan.

void **main**() {

**greet**();

}

void **greet**() {

**print**('Hello!');

}

Pada contoh sederhana di atas fungsi greet() memang belum menghemat banyak kode yang Anda tulis. Namun, apabila Anda memiliki 30 instruksi greet dan ternyata versi terbaru aplikasi Anda memerlukan perubahan teks yang ditampilkan, Anda cukup ubah satu baris kode saja, tak perlu 30 baris kode yang berbeda. Selain itu, jika Anda memiliki kode yang cukup panjang akan lebih baik jika kode tersebut dimasukkan ke dalam fungsi supaya lebih mudah dibaca.

### Function parameters

Pada beberapa kasus fungsi bisa memerlukan input data untuk diproses. Input data ini kita kenal sebagai ***parameter***. Untuk menambahkan parameter ke dalam fungsi, kita bisa memasukkannya ke dalam tanda kurung. Sebuah fungsi bisa menerima nol, satu, atau beberapa parameter.

Contoh penggunaan parameter pada fungsi yang pernah kita lihat adalah pada fungsi print().

1. print('Hello Dart!');

Berikut ini adalah contoh fungsi dengan dua parameter.

void **main**() {

**greet**('Dicoding', 2015);

}

void **greet**(String name, int bornYear) {

  var age = 2023 - bornYear;

**print**('Halo $name! Tahun ini Anda berusia $age tahun');

}

Sebuah fungsi juga bisa menghasilkan output atau mengembalikan nilai. Fungsi yang mengembalikan nilai ditandai dengan definisi return type selain void dan memiliki keyword return. Contohnya seperti berikut.

void **main**() {

  var firstNumber = 7;

  var secondNumber = 10;

**print**(

      'Rata-rata dari $firstNumber & $secondNumber adalah ${**average**(firstNumber, secondNumber)}');

}

double **average**(num num1, num num2) {

  return (num1 + num2) / 2;

}

Jika fungsi hanya memiliki satu baris kode atau instruksi di dalamnya, maka bisa disingkat dengan anotasi =>. Ini juga dikenal dengan nama arrow syntax.

1. double average(num num1, num num2) => (num1 + num2) / 2;
2. void greeting() => print('Hello');

### Optional parameters

Anda memiliki fungsi seperti berikut:

1. void greetNewUser(String name, int age, bool isVerified)

Satu-satunya cara untuk bisa memanggil fungsi di atas adalah dengan cara berikut:

1. greetNewUser('Widy', 20, true);

Namun, Dart mendukung ***optional parameter***, di mana kita tidak wajib mengisi parameter yang diminta oleh fungsi. Untuk bisa membuat parameter menjadi opsional, kita perlu memasukkannya ke dalam kurung siku seperti contoh berikut:

1. void greetNewUser([String name, int age, bool isVerified])

Cara ini disebut dengan ***positional optional parameters***. Dengan optional parameter seperti di atas kita bisa memanggil fungsi seperti berikut:

1. greetNewUser('Widy', 20, true);
2. greetNewUser('Widy', 20);
3. greetNewUser('Widy');
4. greetNewUser();

Setiap parameter yang tidak dimasukkan akan memiliki nilai ***null***. Namun sejak versi 2.12, Dart memiliki fitur ***null safety*** di mana suatu variabel secara default tidak bisa memiliki nilai null. Solusinya, kita bisa menambahkan tanda tanya (?) setelah tipe data supaya bisa menyimpan nilai null atau memberikan nilai parameter default. Nilai ini akan digunakan jika kita tidak memasukkan parameternya.

1. void greetNewUser(**[String? name, int? age, bool isVerified = false]**) {}

Note : Kita akan membahas lebih dalam fitur null safety pada materi “Null Safety”.

Dengan cara ini, urutan parameter masih perlu diperhatikan sehingga jika kita hanya ingin mengisi parameter terakhir, kita perlu mengisi parameter sebelumnya dengan null.

1. greetNewUser(null, null, true);

Untuk mengatasi masalah di atas kita bisa memanfaatkan ***named optional parameters***. Pada opsi ini kita menggunakan kurung kurawal pada parameter.

1. void greetNewUser({String? name, int? age, bool? isVerified})

Dengan cara ini Anda bisa memasukkan parameter tanpa mempedulikan urutan parameter dengan menyebutkan nama parameternya.

1. greetNewUser(name: 'Widy', age: 20, isVerified: true);
2. greetNewUser(name: 'Widy', age: 20);
3. greetNewUser(age: 20);
4. greetNewUser(isVerified: true);

Perhatikan bahwa parameter ini bersifat opsional dan secara default akan bernilai null. Untuk memenuhi null safety, Anda bisa menggunakan cara seperti sebelumnya, atau menandai parameter wajib diisi dengan keyword **required**.

1. void greetNewUser(**{required String name, required int age, bool isVerified = false}**) {}

## Variable Scope

Setelah Anda memisahkan kode Anda ke dalam blok atau fungsi yang terpisah, perlu Anda ketahui bahwa hal tersebut akan mempengaruhi bagaimana suatu variabel digunakan. Setiap variabel memiliki scope atau lingkupnya masing-masing. Sebuah variabel dianggap satu lingkup selama masih berada di satu blok kurung kurawal yang sama. Lingkup ini menentukan bagian kode mana yang dapat membaca dan menggunakan variabel tersebut.

Perhatikan kode berikut ini:

void **main**() {

  var isAvailableForDiscount = true;

  var price = 300000;

  num discount = 0;

  if (isAvailableForDiscount) {

    discount = 10 / 100 \* price;

  }

**print**('You need to pay: ${price - discount}');

}

Pada kode di atas variabel discount masih bisa diakses dari dalam kode if karena masih berada di dalam satu scope fungsi main(). Bagaimana jika Anda ingin memisahkan kode di atas menjadi dua fungsi untuk menghitung diskonnya?

void **main**() {

  var price = 300000;

  var discount = **checkDiscount**(price);

**print**('You need to pay: ${price - discount}');

}

num **checkDiscount**(num price) {

  num discount = 0; // pembuatan variabel baru dengan scope lebih kecil

  if (price >= 100000) {

    discount = 10 / 100 \* price;

  }

  return discount;

}

Variabel discount dideklarasikan pada fungsi checkDiscount() sehingga memiliki scope pada fungsi tersebut saja. Nilainya akan berbeda dengan variabel **discount** pada fungsi main(). Saat variabel dideklarasikan pada suatu fungsi tertentu, ia hanya menjangkau di dalamnya saja karena cakupannya tidak bisa menjangkau ke fungsi lain.

Berbeda cerita ketika Anda mendeklarasikan variabel secara global, yaitu variabel yang dideklarasikan di luar blok kode apa pun. Variabel ini bisa diakses di mana pun selama masih berada di berkas yang sama.

var price = 300000;

void **main**() {

  var discount = **checkDiscount**(price); // variabel price dapat diakses di main()

**print**('You need to pay: ${price - discount}');

}

num **checkDiscount**(num price) {

  num discount = 0;

  if (price >= 100000) {

    // selain itu, ia dapat diakses juga di checkDiscount(),

    discount = 10 / 100 \* price; // bahkan, di level pengondisian if.

  }

  return discount;

}

Bahkan, variabel price dapat memiliki cakupan yang lebih dalam sehingga memungkinkan diakses hingga ke level control flow atau pengondisian **if**. Namun, perlu Anda perhatikan bahwa variabel hanya dapat dipanggil pada cakupan yang ia miliki. Ia tidak dapat diakses jika diluar kemampuannya.

Program di bawah ini adalah salah satu contoh kesalahan dalam memanggil variabel. Ia memanggil variabel discountApplied, tetapi proses deklarasinya terjadi pada perintah berikutnya. Hal ini harus dihindari karena pemanggilan variabel dapat dilakukan setelah proses deklarasi terjadi.

var price = 300000;

void **main**() {

  var discount = **checkDiscount**(price);

**print**('You need to pay: ${price - discount}');

}

num **checkDiscount**(num price) {

  num discount = 0;

  if (!discountApplied) {

    // pemanggilan variabel yang salah sehingga terjadi error

    if (price >= 100000) {

      discount = 10 / 100 \* price;

      var discountApplied =

          true; // proses deklarasi seharusnya terjadi sebelum pemanggilan variabel

    }

  }

  return discount;

}

## Constants & Final

Terkadang kita butuh menyimpan sebuah nilai yang tidak akan pernah berubah selama program berjalan. Variabel telah membantu kita untuk menyimpan nilai dan bisa diakses dengan nama yang deskriptif. ***Constants*** adalah hal baru yang akan kita pelajari dan berguna untuk menyimpan nilai yang tidak akan berubah selama program berjalan.

Sesuai pengertian di atas, kita bisa mendefinisikan nilai yang konstan pada program kita. Salah satu contoh paling mudah yang bisa kita ambil adalah nilai pi = 3.14. Untuk mendefinisikan variabel konstan, gunakanlah keyword const.

1. const pi = 3.14;

Type inference dari Dart akan secara otomatis mendeteksi tipe data dari const pi di atas sebagai double, namun Anda dapat menentukan tipe data secara eksplisit.

1. const num pi = 3.14;

Sehingga pada sebuah aplikasi kalkulator luas lingkaran, kode Anda akan menjadi seperti berikut:

const num pi = 3.14;

void **main**() {

  var radius = 7;

**print**(

      'Luas lingkaran dengan radius $radius = ${**calculateCircleArea**(radius)}');

}

num **calculateCircleArea**(num radius) => pi \* radius \* radius;

Selain const, opsi lain untuk menghindari perubahan nilai variabel setelah diinisialisasi adalah final. Apa bedanya final dan const?

Variabel yang dideklarasikan sebagai const bersifat compile-time constants, artinya nilai tersebut harus sudah diketahui sebelum program dijalankan. Sedangkan final, nilainya masih bisa diinisialisasi ketika runtime atau ketika aplikasi berjalan. Sebagai contoh kode berikut:

void **main**() {

  final firstName = "Achmad";

  final lastName = "Ilham";

  //lastName = 'Angga';       // tidak bisa dilakukan pengubahan nilai

**print**('Hello $firstName $lastName');

}

Kita masih bisa menerima input dan menyimpannya pada variabel firstName dan lastName, namun nilai variabel tersebut tidak bisa diubah setelah diinisialisasi.

Variabel yang nilainya tidak bisa berubah dikenal dengan***immutable variable***. Mutability ini memungkinkan kita terhindar dari bug yang tidak terduga karena terjadi perubahan nilai. Maka dari itu jika Anda yakin bahwa variabel Anda nilainya tetap, gunakanlah const atau final.

Jadi kapan kita harus menggunakan const dan kapan final? Kapan pun memungkinkan, selalu gunakan const karena compile-time constant memiliki performa yang lebih baik dan menggunakan memori lebih sedikit. Jika tidak memungkinkan untuk menggunakan const, gunakan final untuk melindungi variabel agar tidak berubah.

## Null Safety

Sejak versi 2.12, Dart menghadirkan fitur Null Safety. Apakah yang dimaksud dengan null safety dan kenapa kita membutuhkannya?

Sebelum membahas tentang null safety lebih jauh, mari kita ulas kembali terkait variabel dan tipe data.

Seperti yang telah kita bahas sebelumnya, variabel digunakan untuk menyimpan nilai. Variabel String digunakan untuk menyimpan teks, num digunakan untuk menyimpan angka, dsb.

1. String name = 'John Doe';
2. int age = 23;

Namun, kenyataannya suatu variabel juga bisa tidak memiliki nilai. Nilai “tidak ada” pada Dart ini dikenal dengan nilai null. Nilai null bisa terjadi karena kita belum menginisialisasi nilai atau memang secara eksplisit menentukan nilai null pada variabel.

1. String favoriteFood = null;

Lalu, kenapa kita mesti peduli dengan nilai null ini? Nilai null bisa menyebabkan suatu program mengalami kesalahan yang dikenal dengan Null Pointer Exception (NPE). Bahkan, developer yang pertama kali mengenalkan NPE menyebutnya sebagai **“the billion dollar mistake”**.

Bayangkan ada teman yang ingin mentraktir makanan favorit Anda, tetapi karena Anda tidak memiliki makanan favorit, maka ia akan bingung. Begitu pula dengan komputer. Jika nilai variabel null atau “tidak ada”, maka komputer juga akan bingung hingga bisa menyebabkan crash pada program.

Namun, nilai null tidak sepenuhnya buruk. Akan tetap ada kasus tertentu di mana kita membutuhkan nilai null. Contohnya seperti variabel favoriteFood di atas karena tidak semua orang memiliki makanan favorit.

Null safety adalah jawaban dari kasus di atas.

Untuk menggunakan fitur null safety, kita perlu menggunakan versi Dart 2.12. Anda dapat menentukan versi Dart yang digunakan dengan mengubah berkas **pubspec.yaml** yang terdapat dalam project. Pastikan minimal versi sdk yang digunakan adalah versi 2.12.0.

1. environment:
2. sdk: '**>=2.12.0** <3.0.0'

Dengan null safety, secara default sebuah variabel tidak bisa memiliki nilai null, kecuali kita mendeklarasikannya secara eksplisit.

1. int age = null;               // Compile error
2. String? favoriteFood = null;

Tanda tanya (?) di atas menunjukkan bahwa variabel favoriteFood boleh memiliki nilai null (nullable). Sementara variabel age harus memiliki nilai angka dan tidak boleh null (non-nullable).

Kode di atas tidak akan bisa dijalankan karena gagal dalam proses kompilasi. Oleh karena itu, null safety sangat berguna untuk membuat kode menjadi lebih aman karena proses yang melibatkan nilai null dapat dibatasi dan ditemukan lebih awal.

Jika kita memiliki fungsi dengan parameter non-nullable, maka argumen nullable akan terdeteksi sebagai error.

void **main**() {

  String? favoriteFood = null;

**buyAMeal**(favoriteFood); // Compile error

}

void **buyAMeal**(String favoriteFood) {

**print**('I bought a $favoriteFood');

}

Untuk mengakses atau menangani variabel null seperti di atas, ada beberapa cara yang bisa kita lakukan. Pertama, ubah parameter agar dapat menerima nilai null lalu lakukan pengecekan nilai null.

1. void buyAMeal(String? favoriteFood) {
2. if (favoriteFood == null) {
3. print('Bought Nasi Goreng');
4. } else {
5. print('Bought $favoriteFood');
6. }
7. }

Cara kedua yang bisa kita lakukan adalah menggunakan bang operator (!). Dengan operator ini kita memberitahu compiler dan memberikan jaminan bahwa variabel tidak akan bernilai null. Namun, ketika variabel ternyata bernilai null, akan tetap memungkinkan terjadi crash. Jadi, gunakan bang operator ini hanya ketika Anda yakin 100% bahwa variabel tersebut tidak akan bernilai null.

1. void main() {
2. String? favoriteFood = 'Mie Ayam';
4. **buyAMeal(favoriteFood!);**  // disematkan bang operator supaya mengembalikan nilai non-null
5. }
7. void buyAMeal(String favoriteFood) {
8. print('I bought a $favoriteFood');
9. }

Anda dapat mengunjungi tautan berikut jika ingin melihat sumber dari keseluruhan interactive code di atas.