**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:

1. void main() {
2. var greetings = 'Hello Dart!';
3. print(greetings);
4. }

Pada contoh kode di atas kita memasukkan data berupa teks ke dalam variabel. Lalu, bagaimana dengan data numerik atau berupa angka? Tentu saja bisa. Anda cukup menginisialisasi variabel dengan nilai angka.

1. var myAge = 20;
2. print(myAge);

Lalu bagaimana jika kita ingin membuat variabel namun tidak ingin langsung menginisialisasinya? Misalnya Anda ingin menunggu beberapa operasi selesai lalu menginisialisasinya ke dalam variabel.

Proses ini disebut dengan deklarasi variabel. Deklarasi variabel akan menyimpan nama dan ruang di dalam memori tanpa memberikan nilai. Anda dapat menginisialisasi nilai setelah deklarasi sesuai dengan kebutuhan Anda.

1. var myAge;
2. myAge = 20;
3. print(myAge);

Sekarang cobalah *comment* kode inisialisasi variabel di atas lalu jalankan programnya, apa yang terjadi?

Apakah konsol Anda menampilkan *null*? Setiap deklarasi variabel akan memberikan nilai *default* berupa *null*. Ini berarti variabel tersebut belum terinisialisasi atau bisa dibilang variabel Anda tidak memiliki nilai atau *null*.

## 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.

1. var x;  // dynamic
2. x = 7;
3. x = 'Dart is great';
4. 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.

1. var x = 7;  // int
2. x = 'Dart is great'; // Kesalahan assignment
3. print(x);

### 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. Sebelum melihat kode solusi di bawah bisakah Anda menerjemahkan flowchart berikut menjadi kode aplikasi?

### 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.

1. // String -> int
2. var eleven = int.parse('11');
4. // String -> double
5. var elevenPointTwo = double.parse('11.2');
7. // int -> String
8. var elevenAsString = 11.toString();
10. // double -> String
11. var piAsString = 3.14159.toStringAsFixed(2); // String piAsString = '3.14'

## 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.

String singleQuote = 'Ini adalah String';

String doubleQuote = "Ini juga String";

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

print('"What do you think of Dart?" he asked');

Lalu bagaimana jika teks kita memiliki kedua tanda petik ini?

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:

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

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.

print("Windows path: C:\\Program Files\\Dart");

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 $.

var name = 'Messi';

print('Hello $name, nice to meet you.');

print('1 + 1 = ${1 + 1}');

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:

print(r'Dia baru saja membeli komputer seharga $1,000.00');

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.

print('Hi \u2665');

/\*

  output :

    Hi ♥

\*/

## 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.

bool alwaysTrue = true;

var alwaysFalse = false;

var notTrue = !true;

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.

if(true) {

print("It's true");

} else {

print("It's False");

 }

Kita akan segera mempelajari tentang ini pada materi-materi selanjutnya.

## 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. var firstNumber = 4;
2. var secondNumber = 13;
3. var sum = firstNumber + secondNumber;
4. print(sum);
6. /\*
7. Output :
8. 17
9. \*/

Dart mendukung operator aritmatika umum sebagai berikut:

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

1. print(5 + 2);   // int add = 7
2. print(5 - 2);   // int subtract = 3
3. print(5 \* 2);   // int multiply = 10
4. print(5 / 2);   // double divide = 2.5
5. print(5 ~/ 2);  // int intDivide = 2
6. 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.

1. if (2 <= 3) {
2. print('Ya, 2 kurang dari sama dengan 3');
3. } else {
4. print(‘Anda salah’);
5. }
7. /\*
8. Output:
9. Ya, 2 kurang dari sama dengan 3
10. \*/

### 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:

1. if (2 < 3 && 2 + 4 == 5) {
2. print('Untuk mencetak ini semua kondisi harus benar');
3. } else {
4. print('2 kurang dari 3, tapi 2 + 4 tidak sama dengan 5, maka ini akan tampil');
5. }
7. if (false || true || false) {
8. print('Ada satu nilai true');
9. } else {
10. print('Jika semuanya false, maka ini akan tampil');
11. }
13. /\*
14. Output:
15. 2 kurang dari 3, tapi 2 + 4 tidak sama dengan 5, maka ini akan tampil

## Exceptions

Sebuah aplikasi yang sudah berjalan mungkin mengalami eror dan crash. Kondisi eror pada saat aplikasi berjalan (runtime) ini dikenal dengan ***exception***. Ketika exception terjadi aplikasi akan dihentikan dan program setelahnya juga tidak akan dieksekusi.

Salah satu contoh exception yang mungkin terjadi adalah pada aplikasi kalkulator. Di mana menurut prinsip matematika kita tidak bisa membagi bilangan lain dengan bilangan nol (0).

1. var a = 7;
2. var b = 0;
3. print(a ~/ b);

Coba jalankan kode di atas. Anda akan mendapatkan eror seperti berikut:

Unhandled exception:  
IntegerDivisionByZeroException  
#0 int.~/ (dart:core-patch/integers.dart:24:7)  
#1 main (file:///home/dicoding/Playground/dart/Belajar%20Dart/bin/main.dart:24:11)  
#2 \_startIsolate.<anonymous closure> (dart:isolate-patch/isolate\_patch.dart:305:32)  
#3 \_RawReceivePortImpl.\_handleMessage (dart:isolate-patch/isolate\_patch.dart:174:12)

Sebagai developer, tentunya menjadi tugas kita untuk memastikan aplikasi tetap berjalan bagaimana pun kondisinya, termasuk mengatasi ketika terjadi exception. Lantas bagaimana?

Untuk menangani exception, mari gunakan ***try*** dan ***catch***. Pindahkan kode yang berpeluang menimbulkan exception ke dalam blok try.

1. try {
2. var a = 7;
3. var b = 0;
4. print(a ~/ b);
5. }

Kode di atas masih belum lengkap karena di dalam sebuah blok try ada kode yang diasumsikan berpeluang menjadi exception sehingga perlu ditangani. Pada eror sebelumnya telah diketahui bahwa exception yang terjadi adalah IntegerDivisionByZeroException. Sehingga kita bisa memanfaatkan keyword on untuk mengatasi ketika exception tersebut terjadi.

1. try {
2. var a = 7;
3. var b = 0;
4. print(a ~/ b);
5. } on IntegerDivisionByZeroException {
6. print('Can not divide by zero.');
7. }

Nah, kita telah berhasil membuat program kita aman dari crash ketika berusaha membagi bilangan dengan nol. Namun bagaimana dengan exception lainnya yang belum kita ketahui? Pada Dart sendiri telah tersedia beberapa exception seperti IOException, FormatException, dsb. Untuk menangani exception yang tidak diketahui secara spesifik, kita bisa menggunakan keyword catch setelah blok try.

1. try {
2. var a = 7;
3. var b = 0;
4. print(a ~/ b);
5. } catch(e) {
6. print('Exception happened: $e');
7. }

Kode catch membutuhkan satu parameter yang merupakan objek exception. Kita dapat mencetak exception tersebut ke layar untuk menampilkan exception apa yang terjadi. Output kode di atas adalah:

1. Exception happened: IntegerDivisionByZeroException

Selain itu, kita juga dapat menambahkan satu parameter lagi di dalam catch yang merupakan objek stack trace. Dari stack trace ini kita bisa melihat detail exception dan di baris mana exception tersebut terjadi.

1. try {
2. var a = 7;
3. var b = 0;
4. print(a ~/ b);
5. } catch(e, s) {
6. print('Exception happened: $e');
7. print('Stack trace: $s');
8. }
10. /\*
11. Output :
12. Exception happened: IntegerDivisionByZeroException
13. Stack trace: #0      int.~/ (dart:core-patch/integers.dart:24:7)
14. #1      main (file:///home/dicoding/Playground/dart/Belajar%20Dart/bin/main.dart:25:13)
15. #2      \_startIsolate.<anonymous closure> (dart:isolate-patch/isolate\_patch.dart:305:32)
16. #3      \_RawReceivePortImpl.\_handleMessage (dart:isolate-patch/isolate\_patch.dart:174:12)
17. \*/

Blok catch dapat digabungkan dengan on. Catch akan menangkap apabila tidak ada exception yang memenuhi blok on yang terpasang.

### Finally

Selain try, on, dan catch, ada satu blok lagi yang ada dalam mekanisme exception handling, yaitu finally. Blok finally akan tetap dijalankan tanpa peduli apa pun hasil yang terjadi pada blok try-catch.

1. try {
2. var a = 7;
3. var b = 0;
4. print(a ~/ b);
5. } catch(e, s) {
6. print('Exception happened: $e');
7. print('Stack trace: $s');
8. } finally {
9. print('This line still executed');
10. }

## Latihan Aplikasi Konversi Suhu

Katakanlah Anda diundang untuk menghadiri acara konferensi developer di Amerika, namun Anda kebingungan karena ternyata Amerika menggunakan standar suhu Fahrenheit. Tentu jika Anda memiliki aplikasi yang dapat mengonversi suhu dari Fahrenheit menjadi Celsius, akan sangat berguna, bukan?

Sebelum masuk ke pembuatan aplikasi, tentu kita harus tahu rumus konversi suhu terlebih dulu. Alur aplikasi konversi suhu kita akan seperti berikut:

1. Menerima input suhu fahrenheit dari pengguna.
2. Melakukan konversi sesuai rumus.
3. Menampilkan hasil konversi.

Seperti inilah jika proses tersebut dituangkan ke dalam flowchart:

1. Pertama, buat project baru pada IDE Anda lalu berikan nama, misalnya Konversi Suhu.
2. Selanjutnya kita perlu menerima input dari pengguna dan jangan lupa untuk menampilkan informasi apa yang perlu diinputkan. Hapus semua kode pada berkas main.dartlalu tambahkan kode berikut:
   1. import 'dart:io';
   3. void main() {
   4. stdout.write('Masukkan suhu dalam Fahrenheit: ');
   5. var fahrenheit = num.parse(stdin.readLineSync()!);
   6. }
3. Lakukan konversi dengan memasukkan rumus konversi suhu.
   1. var celsius = (fahrenheit - 32) \* 5 / 9;
4. Terakhir tampilkan hasil konversi Anda ke konsol.
   1. print('$fahrenheit derajat Fahrenheit = $celsius derajat celsius');
5. Jalankan dan uji apakah aplikasi Anda telah berjalan dengan sesuai.

#### Challenge

Sebagai tantangan, buatlah aplikasi Anda dapat mendukung konversi suhu lain seperti Reamur, Kelvin, dll.

## 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:

1. void main() {
2. var isAvailableForDiscount = true;
3. var price = 300000;
4. var discount = 0;
5. if (isAvailableForDiscount) {
6. discount = 50000;
7. }
8. print('You need to pay: ${price - discount}');
9. }

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?

1. void main() {
2. var price = 300000;
3. var discount = checkDiscount(price);
4. print('You need to pay: ${price - discount}');
5. }
7. num checkDiscount(num price) {
8. num discount = 0;
9. if (price >= 100000) {
10. discount = 10 / 100 \* price;
11. }
13. return discount;
14. }

Variabel discount dideklarasikan pada fungsi checkDiscount() sehingga memiliki scope pada fungsi tersebut dan menyebabkan eror pada fungsi main(). Maka untuk mengatasinya kita tetap perlu membuat variabel di kedua fungsi.

Selain berada dalam lingkup fungsi, suatu variabel juga bisa menjadi variabel 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.

1. var price = 300000;
3. void main() {
4. var discount = checkDiscount(price);
5. print('You need to pay: ${price - discount}');
6. }
8. num checkDiscount(num price) {
9. num discount = 0;
10. if (price >= 100000) {
11. discount = 10 / 100 \* price;
12. }
14. return discount;
15. }

Variabel juga dapat memiliki scope yang sespesifik mungkin hingga ke level control flow.

1. num checkDiscount(num price) {
2. num discount = 0;
3. if (!discountApplied) { // Error
4. if (price >= 100000) {
5. discount = 10 / 100 \* price;
6. var discountApplied = true;
7. }
8. }
10. return discount;
11. }

## 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:

1. const num pi = 3.14;
3. void main() {
4. var radius = 7;
5. print('Luas lingkaran dengan radius $radius = ${calculateCircleArea(radius)}');
6. }
8. 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:

1. final firstName = stdin.readLineSync();
2. final lastName = stdin.readLineSync();
4. //  lastName = 'Dart'; Tidak bisa mengubah nilai
6. 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.

**Rangkuman Materi**

Selamat! Anda telah mempelajari hal-hal fundamental dalam pemrograman. Sebelum Anda lanjut ke sub-modul berikutnya, berikut adalah beberapa rangkuman dari sub-modul yang sudah dipelajari:

* Comments untuk menuliskan komentar atau penjelasan terhadap kode yang ditulis.
* Terdapat tiga jenis komentar yang tersedia pada Dart:
  + *Single-line comment*atau komentar satu baris yang diawali dengan tanda // dan berakhir pada akhir baris.
  + *Multi-line comment* di mana kita bisa menuliskan beberapa baris komentar yang diawali dengan tanda /\* dan diakhiri dengan \*/.
  + *Documentation comment* adalah kombinasi *single-line*dan *multi-line* yang diawali dengan /// atau /\*\* dan diakhiri dengan /// atau \*\*/.
* Instruksi atau perintah kode yang diawali dengan *comment*akan dilewati ketika proses kompilasi, sehingga tidak akan memengaruhi alur program yang ditulis.
* Variabel digunakan untuk menyimpan nilai. Kita bisa definisikan variabel secara implisit menggunakan *keyword*var.
* Dart mendukung banyak tipe data yang bisa digunakan. Jenis tipe data yang didukung oleh Dart, antara lain:
  + int, tipe data yang dapat menampung bilangan bulat.
  + double, tipe data yang dapat menampung bilangan desimal.
  + num, tipe data yang dapat menampung bilangan bulat (int) dan bilangan desimal (double).
  + bool, tipe data yang hanya memiliki dua nilai, yaitu *true*dan *false*.
  + String, tipe data yang dapat menyimpan teks yang terdiri dari 0 atau beberapa karakter.
  + Collection, tipe data yang dapat menyimpan beberapa objek dalam satu variabel.
  + dynamic, tipe data yang dapat menyimpan tipe data apapun.
* Kita dapat menggabungkan pernyataan boolean atau membaliknya dengan operator logika. Terdapat 3 (tiga) operator logika yang bisa digunakan, meliputi:
  + Operator OR (||), operator yang akan mengembalikan nilai true jika salah satu ekspresi bernilai true.
  + Operator AND (&&), operator yang akan mengembalikan nilai true jika semua ekspresi bernilai true.
  + Operator NOT (!), operator yang melakukan negasi pada ekspresi yang diberikan.
* Function digunakan sebagai mini program atau blok kode yang dapat menghasilkan output dari input tertentu.
* Variabel konstan (const dan final) berguna untuk menyimpan nilai yang tidak akan berubah selama program berjalan.
* Variabel dengan keyword const bersifat compile-time constants, artinya nilai tersebut harus sudah diketahui sebelum program dijalankan.
* Variabel dengan keyword final memiliki sifat di mana nilainya masih bisa diinisialisasi ketika aplikasi berjalan.
* Fitur null safety dari Dart membantu membatasi penggunaan nilai null yang rawan menyebabkan crash pada aplikasi jika tidak ditangani dengan baik.

Pada modul ini juga Anda telah membuat sebuah aplikasi sederhana untuk mengonversi suhu. Apakah Anda siap untuk membuat aplikasi yang lebih kompleks lagi?

Jika masih ada konsep yang belum Anda pahami, jangan ragu untuk membaca kembali modul. Anda juga dapat membuat pertanyaan pada forum diskusi.

Pada modul selanjutnya, kita akan membahas bagaimana mengatur alur program agar sesuai yang kita harapkan.