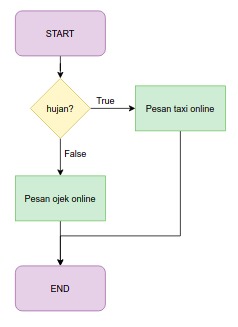
**Pengenalan Control Flow**

Setiap hari kita melakukan perhitungan dan perbandingan guna membuat keputusan, apapun itu. Contohnya apakah perlu mencuci kendaraan ketika cuaca sedang cerah? Apa saja transportasi *online* yang bisa dipesan ketika hujan untuk sampai di tempat tujuan?



Sebuah program juga perlu membuat keputusan. Pada modul ini kita akan belajar bagaimana memberikan instruksi bagi komputer untuk mengambil keputusan dari kondisi yang diberikan serta bagaimana melakukan instruksi yang berulang.

## **If and Else**

Ketika mengembangkan sebuah program, kita akan bertemu dengan alur yang bercabang tergantung kepada kondisi yang terjadi. Untuk mengakomodasi dan mengecek sebuah kondisi pada Dart kita menggunakan kata kunci if.

Ekspresi if akan menguji suatu kondisi. Jika hasil ekspresi tersebut bernilai true, maka blok kode di dalamnya akan dijalankan. Sebaliknya, jika bernilai false maka proses yang ditentukan akan dilewatkan.

void **main**() {

  var isRaining = true;

**print**('Prepare before going to office.');

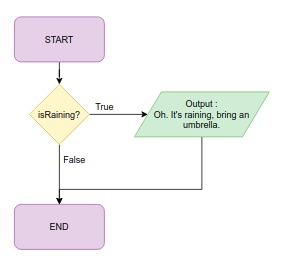
  if (isRaining) {

**print**("Oh. It's raining, bring an umbrella.");

  }

**print**('Going to the office.');

}



Kode di atas akan menampilkan output:

1. Prepare before going to office.
2. Oh. It's raining, bring an umbrella.
3. Going to the office.

Jika Anda mengubah nilai isRaining menjadi false, maka kode di dalam blok if akan dilewatkan dan program Anda tidak akan mengingatkan untuk membawa payung.

Lalu bagaimana jika Anda ingin melakukan operasi lain ketika kondisi bernilai false? Jawabannya adalah dengan menggunakan else. Pada contoh kode berikut kita akan melakukan pengecekan kondisi pada operator perbandingan dan operator logika.

void **main**() {

  var openHours = 8;

  var closedHours = 21;

  var now = 17;

  if (now > openHours && now < closedHours) {

**print**("Hello, we're open");

  } else {

**print**("Sorry, we've closed");

  }

}

Anda juga dapat mengecek beberapa kondisi sekaligus dengan menggabungkan else dan if. Contohnya seperti program konversi nilai berikut:

void **main**() {

  var score = 85;

**print**('Nilai Anda: ${**calculateScore**(score)}');

}

String **calculateScore**(num score) {

  if (score > 90) {

    return 'A';

  } else if (score > 80) {

    return 'B';

  } else if (score > 70) {

    return 'C';

  } else if (score > 60) {

    return 'D';

  } else {

    return 'E';

  }

}

Fitur menarik lain dari Dart adalah ***conditional expressions***. Dengan ini kita bisa menuliskan if-else statement hanya dalam satu baris:

1. // condition ? true expression : false expression
3. var shopStatus = now > openHours ? "Hello, we're open" : "Sorry, we've closed";

Selain itu Dart juga mendukung conditional expressions seperti berikut:

1. // expression1 ?? expression2
2. var buyer = name ?? 'user';

Pada kode di atas jika variabel name tidak bernilai null, maka buyer akan menyimpan nilai dari name. Namun jika bernilai null, buyer akan berisi ‘user’.

Jika ingin melihat sumber keseluruhan interactive code di atas, Anda dapat mengunjungi tautan berikut.

## **For Loops**

Ketika menulis program komputer, akan ada situasi di mana kita perlu melakukan hal sama berkali-kali. Misalnya kita ingin menampilkan semua nama pengguna yang terdaftar di aplikasi kita, atau sesederhana menampilkan angka 1 sampai 10. Tentunya tidak praktis jika kita menulis kode seperti berikut:

1. print(1);
2. print(2);
3. print(3);
4. print(4);
5. print(5);
6. print(6);
7. print(7);
8. print(8);
9. print(9);
10. print(10);

Bagaimana jika kita perlu menampilkan angka 1 sampai 100?

Dart memiliki banyak opsi untuk melakukan perulangan kode, salah satunya adalah for. For cocok digunakan pada kondisi perulangan di mana kita membutuhkan variabel indeks dan tahu berapa kali perulangan yang kita butuhkan. Sebagai contoh jika kita ingin menampilkan angka 1 sampai 100, kita bisa menuliskan seperti berikut:

void **main**() {

  for (int i = 1; i <= 100; i++) {

**print**(i);

  }

}

Lebih ringkas bukan? Terdapat tiga bagian utama dalam sintaks for di atas:

* Pertama, variabel *index* yang seringkali diberi nama i yang berarti *index*. Pada variabel ini kita menginisialisasi nilai awal dari perulangan yang kita lakukan.
* Kedua, operasi perbandingan. Pada bagian ini komputer akan melakukan pengecekan kondisi apakah perulangan masih perlu dilakukan. Jika bernilai true maka kode di dalam blok for akan dijalankan.
* Ketiga, *increment/decrement*. Di sini kita melakukan penambahan atau pengurangan variabel *index*. Jadi pada contoh di atas variabel indeks akan ditambah dengan 1 di setiap akhir perulangan.

Jika dituliskan dalam bentuk *pseudocode*, maka kode di atas bisa dimaknai dengan “Jika i kurang dari sama dengan 100, maka jalankan kode berikut.”

## **While and do-while**

Metode lain untuk melakukan perulangan adalah dengan while. Sama seperti for, instruksi while mengevaluasi ekspresi boolean dan menjalankan kode di dalam blok while ketika bernilai true.

Untuk menampilkan angka 1 sampai 100 dengan while kita bisa menulis kode seperti berikut:

void **main**() {

  var i = 1;

  while (i <= 100) {

**print**(i);

    i++;

  }

}

Bisa dilihat pada kode di atas bahwa perulangan dengan while tidak memiliki ketergantungan dengan variabel index seperti pada for loop. Karena itu, meskipun while dapat melakukan perulangan yang sama dengan for, while lebih cocok digunakan pada kasus di mana kita tidak tahu pasti berapa banyak perulangan yang diperlukan.

Bentuk lain dari while adalah perulangan do-while.

void **main**() {

  var i = 1;

  do {

**print**(i);

    i++;

  } while (i <= 100);

}

Kondisi pada while akan dievaluasi sebelum blok kode di dalamnya dijalankan, sedangkan do-while akan mengevaluasi boolean expression setelah blok kodenya dijalankan. Ini artinya kode di dalam do-while akan dijalankan setidaknya satu kali.

### **Infinite loops**

Ketika menerapkan perulangan pada program kita, ada satu kondisi yang perlu kita hindari yaitu ***infinite loop***. Infinite loop atau endless loop adalah kondisi di mana program kita stucked di dalam perulangan. Ia akan berjalan terus hingga menyebabkan crash pada aplikasi dan komputer kecuali ada intervensi secara eksternal, seperti mematikan aplikasi.

Kode berikut ini adalah contoh di mana kondisi infinite loop dapat terjadi:

void **main**() {

  var i = 1;

  while (i < 5) {

**print**(i);

  }

}

Dapatkah Anda mencari apa yang salah dari kode di atas sehingga terjadi infinite loop?

Jawabannya adalah karena variabel i selalu bernilai 1. Alhasil kondisi i < 5 akan selalu bernilai true dan akibatnya aplikasi akan terus mencetak 1 ke konsol sehingga mengalami crash.

## **Break and Continue**

Anda memiliki aplikasi yang menyimpan data 20 bilangan prima pertama. Pengguna dapat mencari bilangan prima lalu komputer akan menampilkan urutan berapa bilangan tersebut. Ketika bilangan prima tersebut sudah ditemukan tentunya komputer tidak perlu melanjutkan proses perulangan lagi. Nah, di sinilah kita bisa menggunakan ***break*** untuk menghentikan dan keluar dari proses iterasi.

void **main**() {

  // bilangan prima di bawah 100

  var primeNumbers = [

    2,

    3,

    5,

    7,

    11,

    13,

    17,

    19,

    23,

    29,

    31,

    37,

    41,

    43,

    47,

    53,

    59,

    61,

    67,

    73,

    79,

    83,

    89,

    97

  ];

  var searchNumber = 13;

**print**('Bilangan prima di antara 1-100: $searchNumber');

  for (int i = 0; i < primeNumbers.length; i++) {

    if (searchNumber == primeNumbers[i]) {

**print**('$searchNumber adalah bilangan prima ke-${i + 1}');

      break;

    }

**print**('$searchNumber != ${primeNumbers[i]}');

  }

}

Ketika kode di atas dijalankan, proses iterasi akan dihentikan ketika angka yang diinputkan pengguna sama dengan bilangan prima yang dicari.

1. Bilangan prima di antara 1-100: 13
2. 13 != 2
3. 13 != 3
4. 13 != 5
5. 13 != 7
6. 13 != 11
7. 13 adalah bilangan prima ke-6

Keyword lain yang berguna pada proses perulangan adalah ***continue***. Dengan continue kita bisa melewatkan proses iterasi dan lanjut ke proses iterasi berikutnya. Misalnya Anda ingin menampilkan angka 1 sampai 10 kecuali angka kelipatan 3. Anda dapat menuliskannya seperti berikut:

void **main**() {

  for (int i = 1; i <= 10; i++) {

    if (i % 3 == 0) {

      continue;

    }

**print**(i);

  }

}

## **Switch and Case**

Sebelumnya kita telah mempelajari bagaimana mengondisikan logika komputer dengan menggunakan if. Namun, bagaimana jika ada banyak kondisi yang perlu dicek menggunakan if? Tentu akan membingungkan dan kode kita pun jadi sulit dibaca.

Dart mendukung statement switch untuk melakukan pengecekan banyak kondisi dengan lebih mudah dan ringkas.

1. switch (variable/expression) {
2. case value1:
3. // do something
4. break;
5. case value2:
6. // do something
7. break;
8. ...
9. ...
10. default:
11. // do something else
12. }

Tanda kurung setelah keyword switch berisi variabel atau ekspresi yang akan dievaluasi. Kemudian untuk setiap kondisi yang mungkin terjadi kita masukkan keyword case diikuti dengan nilai yang valid. Jika kondisi pada case sama dengan variabel pada switch, maka blok kode setelah titik dua (:) akan dijalankan. Keyword break digunakan untuk keluar dari proses switch. Terdapat satu case bernama default yang memiliki fungsi yang sama dengan keyword else pada control flow if-else. Jika tidak ada nilai yang sama dengan variabel pada switch maka blok kode ini akan dijalankan.

Berikut ini adalah contoh aplikasi kalkulator yang menerapkan switch-case.

void **main**() {

  final firstNumber = 13;

  final secondNumber = 18;

  final operator = "+";

  switch (operator) {

    case '+':

**print**(

          '$firstNumber $operator $secondNumber = ${firstNumber + secondNumber}');

      break;

    case '-':

**print**(

          '$firstNumber $operator $secondNumber = ${firstNumber - secondNumber}');

      break;

    case '\*':

**print**(

          '$firstNumber $operator $secondNumber = ${firstNumber \* secondNumber}');

      break;

    case '/':

**print**(

          '$firstNumber $operator $secondNumber = ${firstNumber / secondNumber}');

      break;

    default:

**print**('Operator tidak ditemukan');

  }

}

## **Pengenalan Collections**

Kita telah mempelajari tentang tipe data dan control flow. Lanjut, kini kita akan menghadapi masalah yang lebih kompleks. Untuk bisa memecahkannya secara efisien, kita membutuhkan struktur data yang lebih canggih.

Selain string, number, dan boolean, Dart masih memiliki tipe data lain yang dapat menyimpan banyak data sekaligus yang dalam istilah pemrograman dikenal sebagai ***collections***. Collections merupakan sebuah objek yang bisa menyimpan kumpulan objek lain. Contoh collections pada Dart antara lain List, Set, dan Map.

## List

List sesuai namanya dapat menampung banyak data ke dalam satu objek. Dalam kehidupan sehari-hari kita menggunakan list untuk menyimpan daftar belanja, nomor telepon, dsb. Begitu pula dengan Dart kita bisa menyimpan bermacam-macam tipe data seperti string, number, dan boolean. Cara penulisannya pun sangat mudah. Perhatikan saja contoh berikut:

List<int> numberList = [1, 2, 3, 4, 5];

Kode di atas adalah contoh dari satu objek List yang berisi kumpulan data dengan tipe integer. Karena kompiler bisa mengetahui tipe data yang ada dalam sebuah objek List, maka tak perlu kita menuliskannya secara eksplisit.

var numberList = [1, 2, 3, 4, 5];

var stringList = ['Hello', 'Dicoding', 'Dart'];

Sesuai contoh di atas, kita mendefinisikan tipe data yang bisa dimasukkan ke dalam List di dalam tanda kurung siku ([ ]). Sama seperti variabel, jika kita tidak mendefinisikan nilai secara eksplisit ke dalam List, maka List akan menyimpan tipe dynamic atau bisa menyimpan semua tipe data.

void **main**() {

  List dynamicList = [1, 'Dicoding', true];

**print**(dynamicList.runtimeType);

}

Ketika bermain dengan sebuah List, tentunya ada saat di mana kita ingin mengakses posisi tertentu dari List tersebut. Untuk melakukannya, kita bisa menggunakan fungsi indexing seperti berikut:

1. void main() {
2. List dynamicList = [1, 'Dicoding', true];
3. print(dynamicList[1]);
4. }
6. // Output: Dicoding

Perhatikan kode di atas. Fungsi indexing ditandai dengan tanda [ ]. Jika Anda mengira bahwa konsol akan menampilkan angka 1, tebakan Anda kurang tepat. Dalam sebuah List, indeks dimulai dari 0. Maka ketika kita akan mengakses data pada dynamicList yang berada pada indeks ke-1, artinya data tersebut merupakan data pada posisi ke-2. Jadi data yang akan ditampilkan pada konsol adalah ***Dicoding***.

Lalu apa yang akan terjadi jika kita berusaha menampilkan item dari List yang berada di luar dari ukuran List tersebut? Sebagai contoh, Anda ingin mengakses indeks ke-3 dari dynamicList:

1. print(dynamicList[3]);

Hasilnya adalah eror! Kompiler akan memberitahukan bahwa perintah itu tidak bisa dijalankan. Berikut pesan eror yang akan muncul:

Unhandled exception:  
RangeError (index): Invalid value: Not in range 0..2, inclusive: 3

Pesan di atas memberitahu kita bahwa List telah diakses dengan indeks ilegal. Ini akan terjadi jika indeks yang kita inginkan negatif atau lebih besar dari atau sama dengan ukuran List tersebut.

Masih ingat looping? Untuk menampilkan seluruh item dari list kita bisa memanfaatkan looping. Contohnya perhatikan kode berikut:

1. void main() {
2. List<String> stringList = ["Hello", "Dicoding", "Dart"];
3. for (int i = 0; i < stringList.length; i++) {
4. print(stringList[i]);
5. }
6. }
8. // Output:
9. // Hello
10. // Dicoding
11. // Dart

Pada kode di atas kita memanfaatkan perulangan sebanyak jumlah data di dalam list untuk mencetak data yang ada di dalam list. Banyaknya data di dalam list bisa kita panggil melalui properti .length.

Selain itu kita juga bisa menggunakan fungsi foreach untuk menampilkan data di dalam list.

void **main**() {

  List<String> stringList = ["Hello", "Dicoding", "Dart"];

  stringList.**forEach**((s) => **print**(s));

}

Mekanisme di atas dikenal sebagai ***lambda*** atau **anonymous function**. Kita akan mempelajarinya lebih dalam pada modul yang akan datang.

Sejauh ini kita baru belajar menginisialisasikan dan mengakses data dari sebuah List. Pastinya Anda bertanya, “Bagaimana kita memanipulasi data pada List tersebut?” Nah, untuk menambahkan data ke dalam list, kita bisa menggunakan fungsi add().

void **main**() {

  List<String> stringList = ["Hello", "Dicoding", "Dart"];

  // Menambahkan data di akhir list.

  stringList.**add**('Flutter');

**print**(stringList);

}

Fungsi add ini akan menambahkan data di akhir list. Sehingga ketika dicetak, konsol akan menampilkan data berikut:

1. [Hello, Dicoding, Dart, Flutter]

Lalu bagaimana jika kita ingin menambahkan data namun tidak di akhir List? Jawabannya adalah dengan fungsi ***insert***. Di dalam fungsi insert kita perlu memasukkan 2 parameter, yaitu indeks list dan data yang akan dimasukkan.

1. void main() {
2. List<String> stringList = ["Hello", "Dicoding", "Dart", "Flutter"];
3. // Menambahkan data di indeks ke-0.
4. stringList.insert(0, 'Programming');
5. print(stringList);
6. }
8. /// Output:
9. /// [Programming, Hello, Dicoding, Dart, Flutter]

Untuk mengubah nilai di dalam list, kita bisa langsung menginisialisasikan nilai baru sesuai indeks yang diinginkan.

1. stringList[1] = 'Application';

Sedangkan untuk menghapus data terdapat beberapa fungsi ***remove*** yang bisa kita gunakan, antara lain:

void **main**() {

  List<String> stringList = [

    "Programming",

    "Hello",

    "Dicoding",

    "Dart",

    "Flutter"

  ];

  // Menghapus list dengan nilai Programming

  stringList.**remove**('Programming');

  // Menghapus list pada index ke-1

  // stringList.removeAt(1);

  // Menghapus data list terakhir

  // stringList.removeLast();

  // Menghapus list mulai index ke-0 sampai ke-1 (indeks 2 masih dipertahankan)

  // stringList.removeRange(0,2);

**print**(stringList);

}

### Spread Operator

Dart memiliki fitur menarik untuk menambahkan banyak nilai ke dalam List dengan cara yang singkat, yaitu spread operator.

Sesuai namanya “spread”, fitur ini digunakan untuk menyebarkan nilai di dalam collections menjadi beberapa elemen. Spread operator dituliskan dengan tiga titik (...). Mari kita lihat contoh kode berikut:

1. void main() {
2. var favorites = ['Seafood', 'Salad', 'Nugget', 'Soup'];
3. var others = ['Cake', 'Pie', 'Donut'];
4. var allFavorites = [favorites, others];
5. print(allFavorites);
6. }
8. /// Output:
9. /// [[Seafood, Salad, Nugget, Soup], [Cake, Pie, Donut]]

Nilai List tidak akan tergabung. Alih-alih menggabungkan nilainya, variabel allFavorites menjadi List yang menampung dua List di dalamnya. Nah, dengan spread operator kita dapat menggabungkan kedua List di atas:

void **main**() {

  var favorites = ['Seafood', 'Salad', 'Nugget', 'Soup'];

  var others = ['Cake', 'Pie', 'Donut'];

  var allFavorites = [...favorites, ...others];

**print**(allFavorites);

}

Yup, dengan spread operator kita dapat memasukkan masing-masing elemen di dalam List ke dalam List lainnya.

Untuk mengatasi List yang bisa bernilai null, kita dapat menggunakan null-aware spread **operator** (...?) seperti berikut:

void **main**() {

  List<dynamic>? list;

  List<dynamic>? list2 = [0, ...?list];

**print**(list2);

}

## **Set**

Selanjutnya kita akan membahas jenis collection yang kedua, yaitu ***Set***. Set merupakan sebuah collection yang hanya dapat menyimpan nilai yang unik. Ini akan berguna ketika Anda tidak ingin ada data yang sama alias duplikasi dalam sebuah collection. Kita bisa mendeklarasikan Set dengan beberapa cara berikut:

void **main**() {

  Set<int> anotherSet = new Set.**from**([1, 4, 6, 4, 1]);

**print**(anotherSet);

}

Perhatikan kode di atas. Di sana terdapat beberapa angka yang duplikat, yaitu angka 1 dan 4. Silakan tampilkan pada konsol dan lihat hasilnya.

1. print(anotherSet);
3. // Output: {1, 4, 6}

Secara otomatis Set akan membuang angka yang sama, sehingga hasilnya adalah {1, 4, 6}.

Untuk menambahkan data ke dalam Set kita dapat memanfaatkan fungsi add() atau addAll()

void **main**() {

  var numberSet = {1, 4, 6};

  // Menambahkan data ke dalam Set.

  numberSet.**add**(6);

  numberSet.**addAll**({2, 2, 3});

**print**(numberSet);

}

Fungsi add akan menambah satu item ke dalam Set, sementara addAll digunakan untuk menambahkan beberapa item sekaligus. Nilai yang duplikat akan diabaikan.

Lalu gunakan fungsi remove() untuk menghapus objek di dalam set yang diinginkan.

1. void main() {
2. var numberSet = {1, 4, 6, 2, 3};
3. // Menghapus nilai 3 dari Set.
4. numberSet.remove(3);
5. print(numberSet);
6. }
8. /// Output:
9. /// {1, 4, 6, 2}

Kode di atas akan menghapus nilai 3 di dalam Set, bukan indeks ke-3.

Kemudian untuk menampilkan data pada indeks tertentu, gunakanlah fungsi elementAt().

1. void main() {
2. var numberSet = {1, 4, 6, 2, 3};
3. // Mendapatkan data Set pada indeks ke-2
4. print(numberSet.elementAt(2));
5. }
7. /// Output:
8. /// 6

Selain itu, Dart juga memiliki fungsi ***union*** dan ***intersection*** untuk mengetahui gabungan dan irisan dari 2 (dua) buah Set. Sebagai contoh:

void **main**() {

  var setA = {1, 2, 4, 5};

  var setB = {1, 5, 7};

  var union = setA.**union**(setB);

  var intersection = setA.**intersection**(setB);

**print**("union: $union");

**print**("intersection: $intersection");

}

## Map

Collection ketiga adalah Map, yakni sebuah collection yang dapat menyimpan data dengan format key-value. Perhatikan contoh berikut:

1. var capital = {
2. 'Jakarta': 'Indonesia',
3. 'London': 'England',
4. 'Tokyo': 'Japan'
5. };

String yang berada pada sebelah kiri titik dua (:) adalah sebuah key, sedangkan yang di sebelah kanan merupakan value-nya. Lalu untuk mengakses nilai dari Map tersebut, kita bisa menggunakan key yang sudah dimasukkan. Misalnya, kita bisa menggunakan key “Jakarta” untuk mendapatkan value “Indonesia”:

void **main**() {

  var capital = {

    'Jakarta': 'Indonesia',

    'London': 'England',

    'Tokyo': 'Japan',

  };

**print**(capital['Jakarta']);

}

Kita dapat menampilkan key apa saja yang ada di dalam Map dengan menggunakan property keys. Sedangkan untuk mengetahui nilai apa saja yang ada di dalam Map kita bisa menggunakan property values.

void **main**() {

  var capital = {

    'Jakarta': 'Indonesia',

    'London': 'England',

    'Tokyo': 'Japan',

  };

  var mapKeys = capital.keys;

**print**("mapKeys: $mapKeys");

  var mapValues = capital.values;

**print**("mapValues: $mapValues");

}

Untuk menambahkan key-value baru ke dalam Map, kita bisa melakukannya dengan cara berikut:

void **main**() {

  var capital = {

    'Jakarta': 'Indonesia',

    'London': 'England',

    'Tokyo': 'Japan',

  };

  capital['New Delhi'] = 'India';

**print**(capital);

}