Akhmad Fauzi (PC-11)

*Mobile Application and Technology*

**LAPORAN MATERI**

**MENGIMPLEMENTASIKAN ALGORITMA PEMOGRAMAN**

“Mengimplementasikan Algoritma Pemograman” yang dibawakan oleh instruktur (Pak Agung), dimana berisi tentang dasar-dasar algoritma dalam pemograman dengan Kotlin.

Dalam membangun suatu program, diperlukan sebuah rancangan dalam menentukan langkah-langkah proses yang akan dijalankan oleh program. Langkah-langkah secara berurutan ini disebut sebagai algoritma yang akan membangun dan menyelesaikan suatu permasalahan. Sebagai dokumentasi, algoritma dapat dituliskan dalam bentuk diagram salah satunya adalah *flowchart. Flowchart* adalah sebuah diagram yang menggambarkan *workflow* atau sebuah proses secara berurutan sesuai dengan simbol-simbol *flowchart*. Berikut ini simbol-simbol utama pada *flowchart* yang ditunjukkan melalui tabel 1.

| **Bentuk Simbol** | **Fungsi** |
| --- | --- |
|  | *Flowline*  Menunjukkan urutan proses pada algoritma menuju arah yang ditunjukkan panah |
|  | *Terminal*  Menunjukkan awal (*start*) dan akhir (*end*) algoritma |
|  | *Process*  Menunjukkan set operasi pada algoritma yang melibatkan perubahan *value, form,* maupun *location data* |
|  | *Input or Output*  Menunjukkan proses yang melibatkan input ataupun output, memasukkan atau menampilkan data |
|  | *Decision*  Menunjukkan *conditional operation* yang biasanya merupakan *yes or no question* |

Tabel 1. Simbol umum yang digunakan pada flowchart

Secara umum, *Control Flow* pada pemograman terdiri dari *sequence* (berurutan), *selection* (*condition*), dan *repetition* (*loops*)*. Control flow sequence* yaitu saat program menjalankan *codes* yang ada di dalam fun main() secara berurutan. *Selection* yaitu saat program mengalami percabangan dan memilih berdasarkan kondisi yang telah didefinisikan, bagian ini merupakan fungsi yang dibangun oleh *if expression* dan *when* (seperti *switch* pada bahasa C)yang dicontohkan melalui gambar 1. Sedangkan *repetition* atau *loops* yaitu sat program mengalami perulangan di dalam suatu program hingga batas yang telah ditentukan, bagian ini merupakan fungsi yang dibangun oleh *do*-*while command*.

***1. Percabangan if***

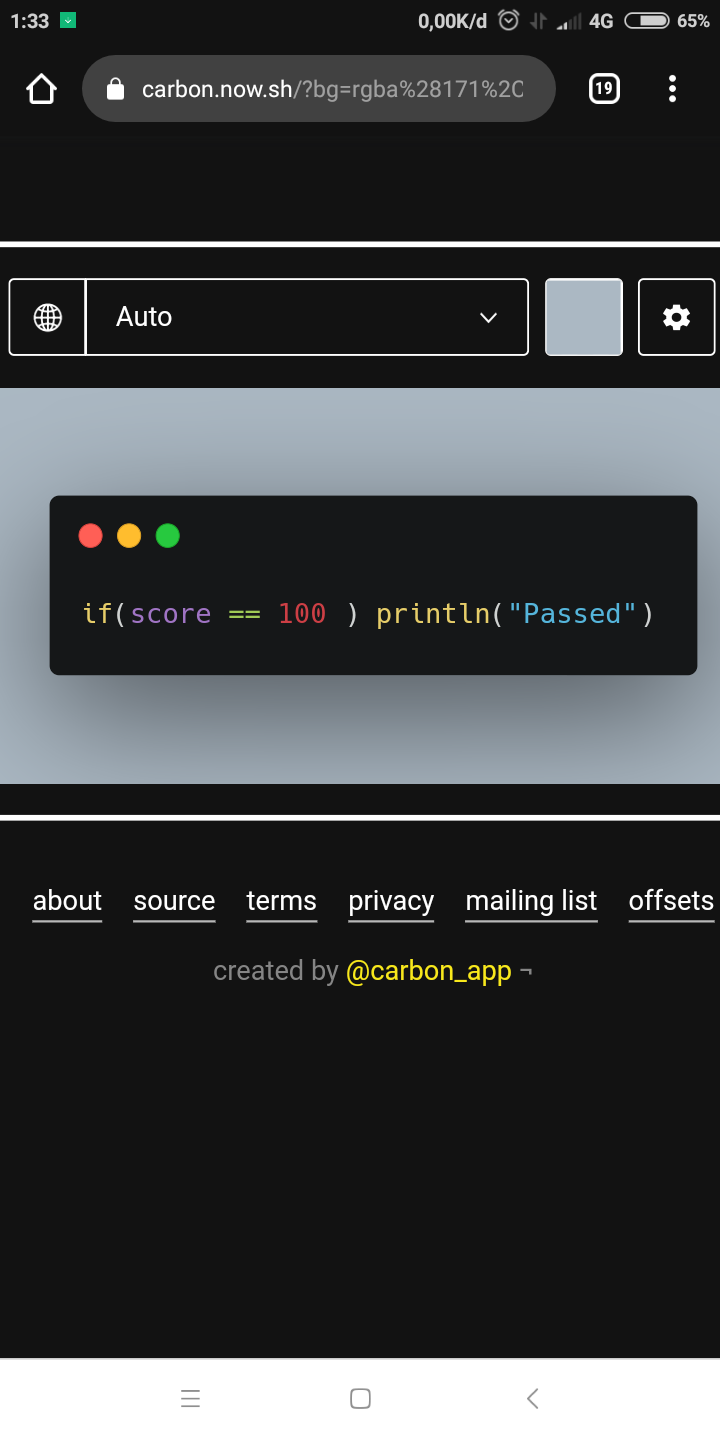
Bentuk percabangan ini adalah bentuk yang paling sederhana. Ia memiliki satu blok kode yang akan dieksekusi apabila kondisinya terpenuhi.

Bentuk dasar:

Dibagian kondisi kita bisa isi dengan sebuah variabel boolean atau pernyataan yang menghasilkan nilai boolen.

contoh: 

Hasilnya:

Apabila hanya terdapat satu baris kode, blok kode yang ada di dalam if boleh tidak dikurung seperti ini:

score == 100 ini adalah sebuah operasi yang akan menghasilkan nilai true dan false (boolean).

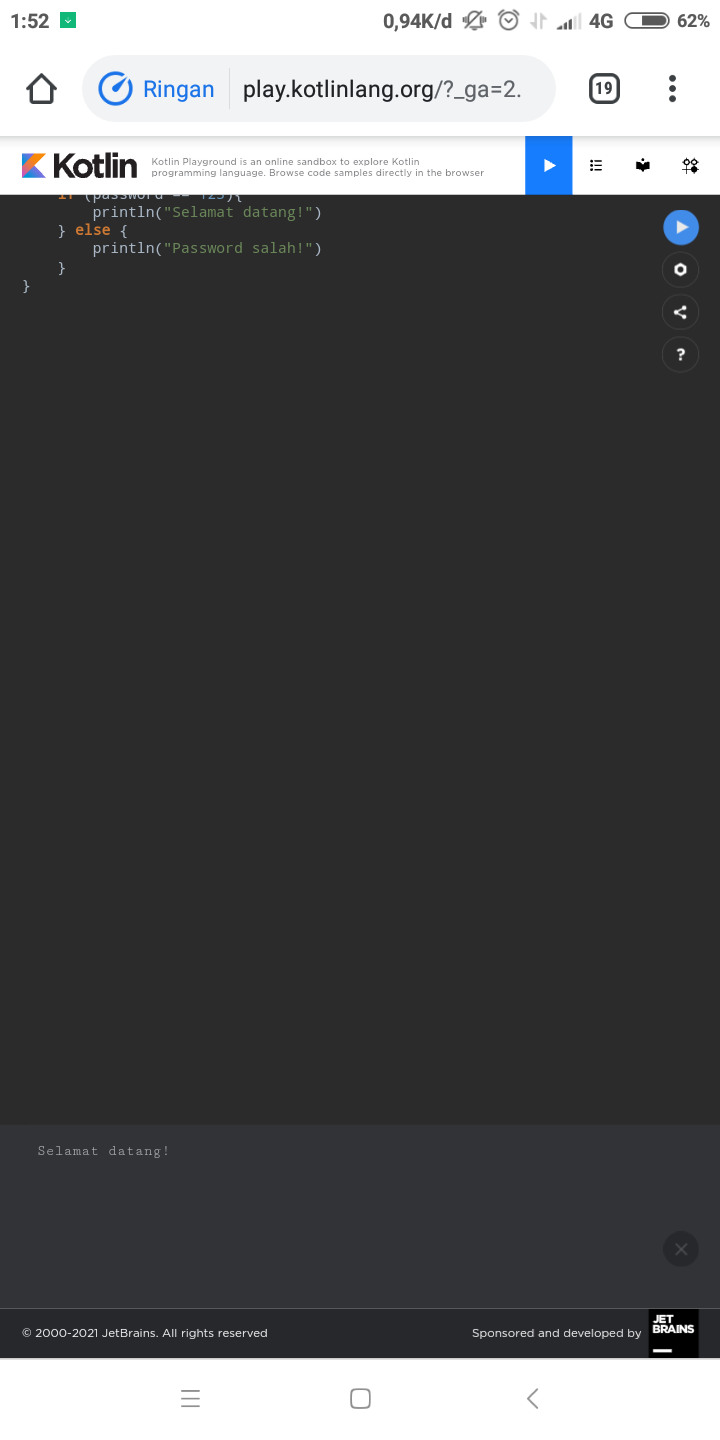
***2. Percabangan if/else***

Percabangan if/else memiliki dua blok kode yang akan dieksekusi. Blok pertama untuk kondisi bernilai true dan blok kedua (else) untuk kondisi bernilai false.

Bentuknya seperti ini:

contoh:



Hasilnya: 

***3. Percabangan if/else if/else***

Percabangan ini memiliki lebih dari dua blok kode dan kondisi.

Bentuknya seperti ini:

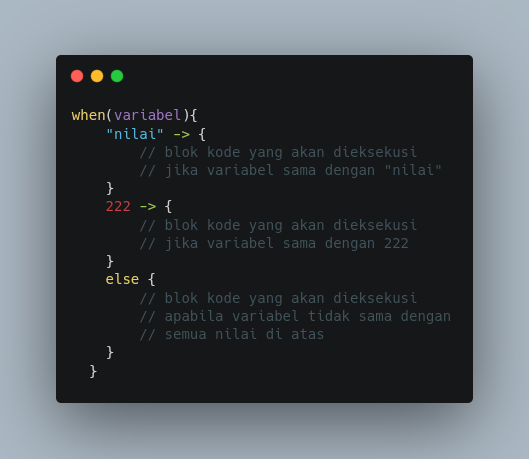
Contoh Programnya:

Hasilnya:



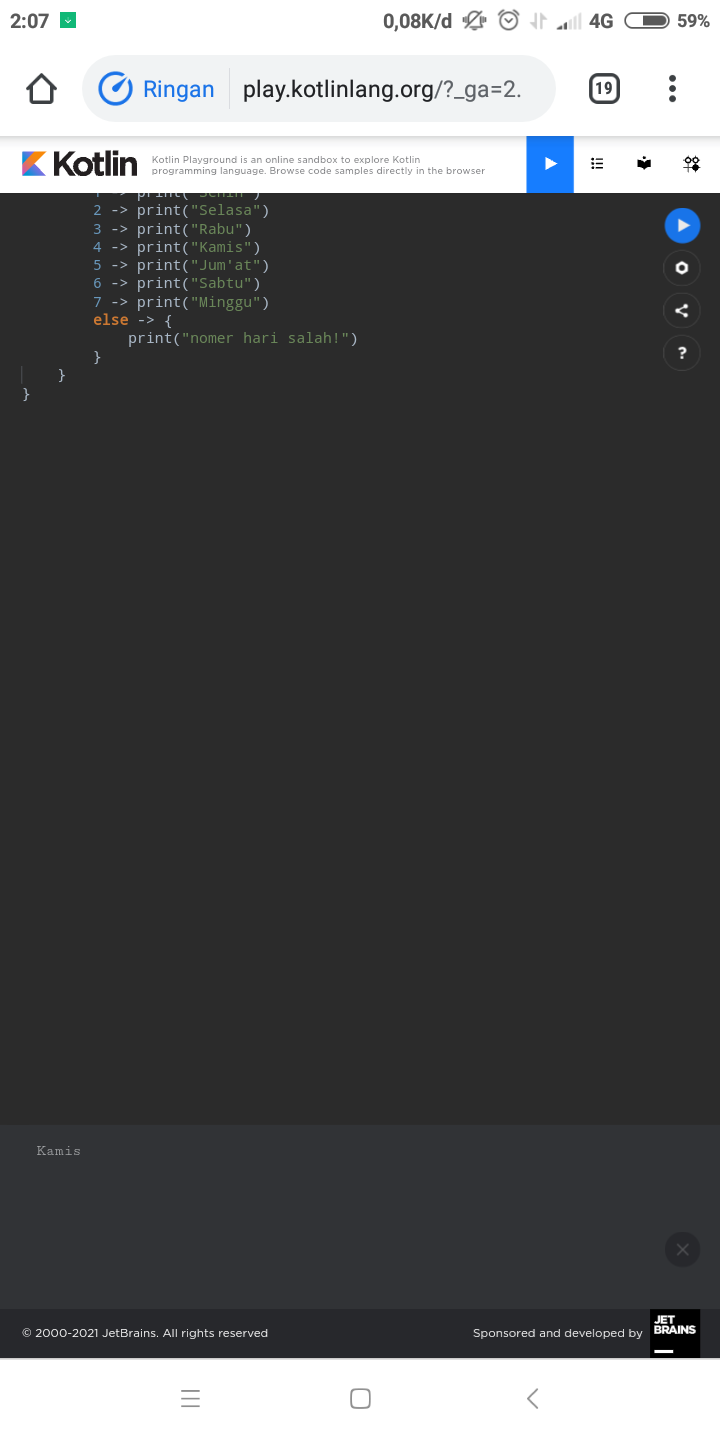
***4. Percabangan When***

Percabangan when sebenarnya adalah percabangan switch/case. Percabangan when sama seperti percabangan if/else if/else yang memiliki banyak blok dan kondisi.

Bentuknya seperti ini:

Contoh Programnya:

Hasinya:



Pada percabangan when, kita juga bisa mengecek nilai variabel dengan range (..).

Contoh Programnya:

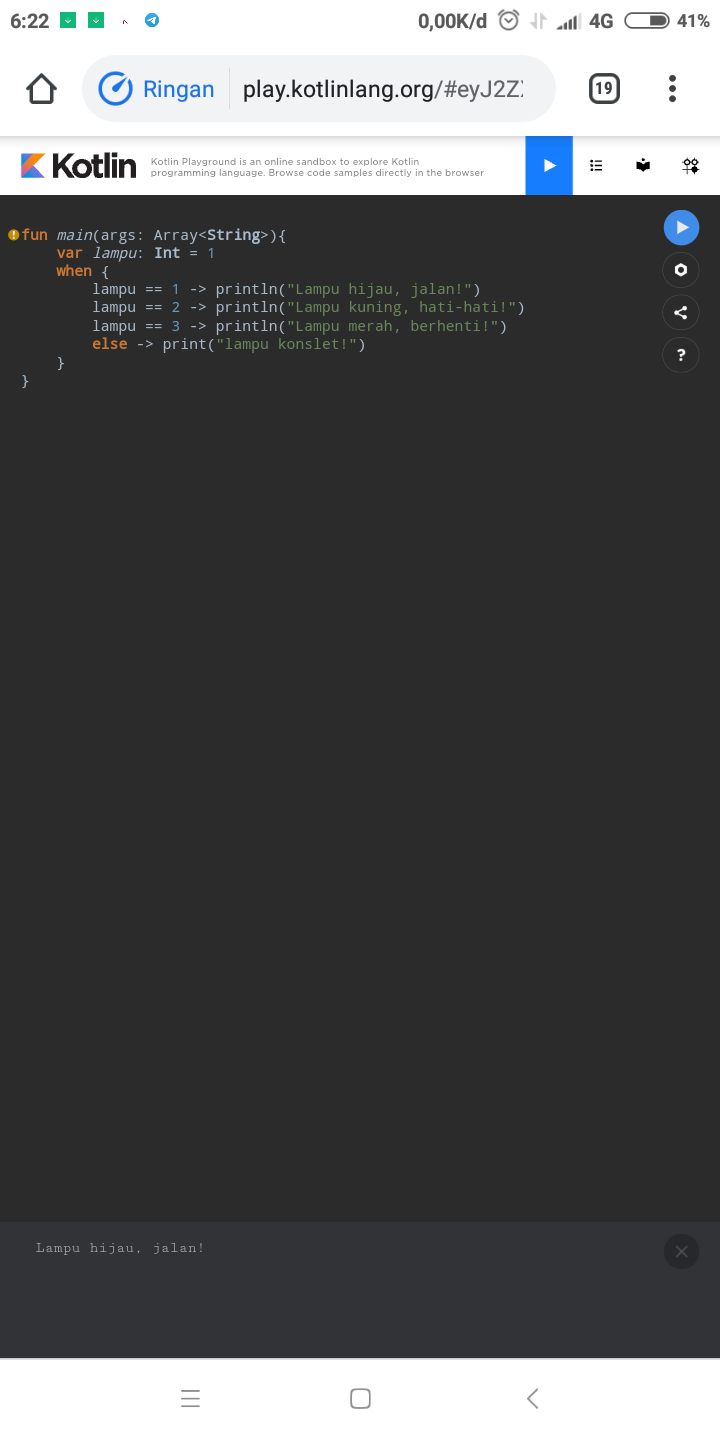
Hasilya:

Arti tanda titik-titik (..) pada contoh program di atas adalah range, Misal 1..100 maka artinya itu adalah nilai 1 sampi 100.

Berikutnya, percabangan when juga bisa tulis seperti ini:

Contoh Programnya:

Hasilnya:

Contoh Program:



Gambar 1. Contoh penggunaan if dan when expression

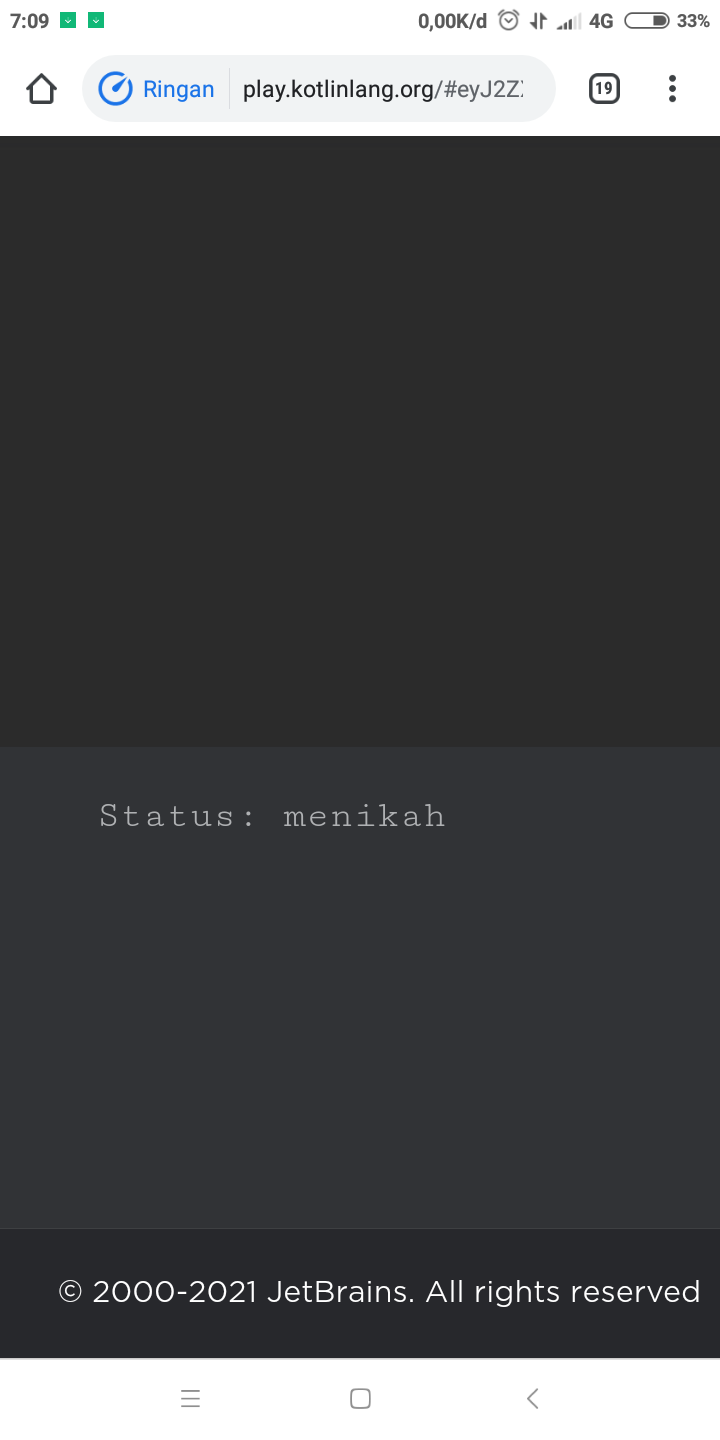
***5. Percabangan dengan Operator Ternary***

Pada kotlin, operator ternary tidak ada. Sebagai gantinya, kita cukup menggunakan ekspresi if/else saja.

Contoh:



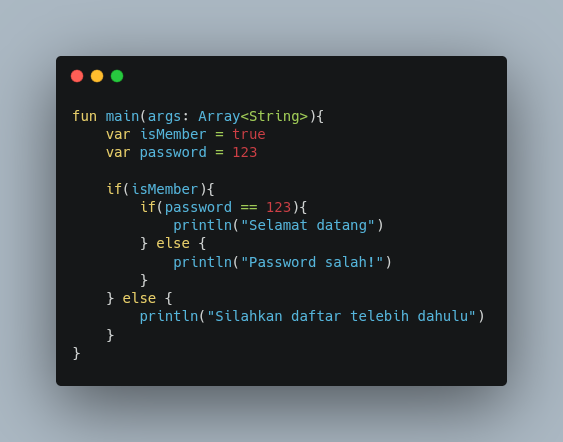
**Hasilnya:**



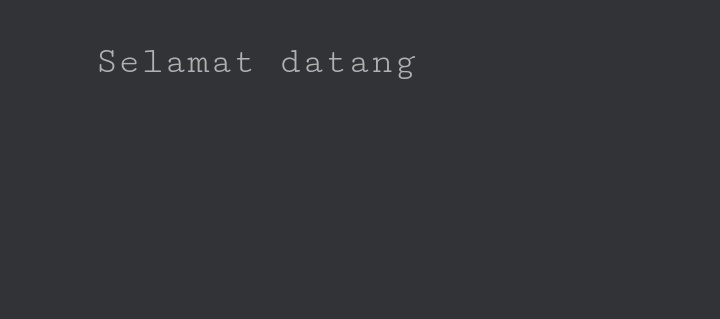
***6. Percabangan Bersarang (Nested)***

**Percabangan bersarang atau nested adalah percabangan di dalam percabangan.**

**Contoh:**



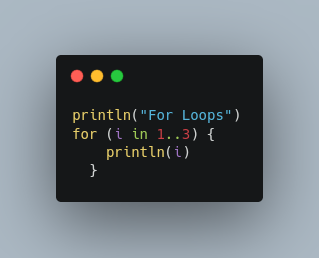
**Hasilnya:**



***For Loops***

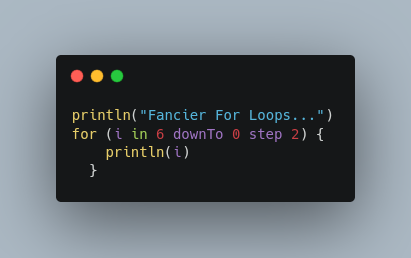
for loop digunakan untuk mengulang (looping) sebuah collection dan juga yang list data yang lain. Pada contoh dibawah, Kita akan menggunakan looping menggunakan range.

Contoh:



Kode diatas akan menampilkan sebuah perulangan(loop) mulai angka 1 hingga 3 menggunakan 1..3 yaitu range, anda juga bisa membuat nya agar lebih epic.

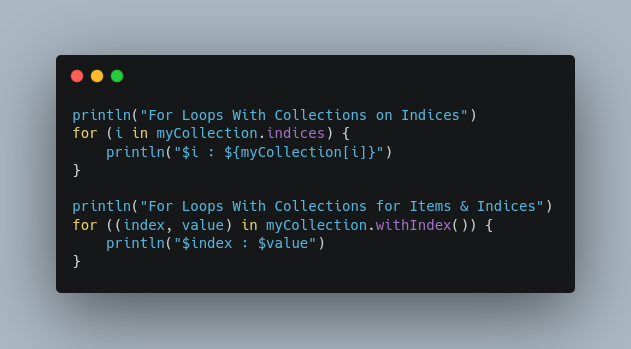
Contoh:



Loop juga dapat digunakan untuk collection, seperti di bawah ini:

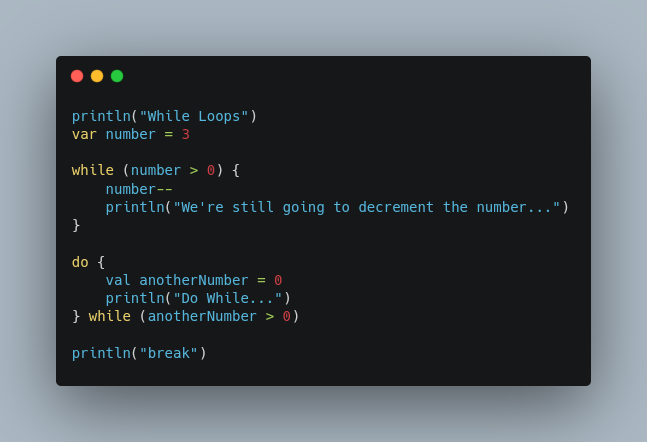


Atau menggunakan indeces withIndex library yang tersedia oleh Kotlin.



***While Loops***

while & do while



while (number > 0)

Yang berarti jika nilai dari object number lebih dari 0, maka nilai dari number akan dikurangi dengan number--.

Dan pada do while, sebenarnya sama saja dengan while hanya saja letak lokasi pengecekannya yang berbeda.

Dan pada do while, sebenarnya sama saja dengan while hanya saja letak lokasi pengecekannya yang berbeda.

Untuk pengecekannya berada di while yang dimana jika anotherNumbersudah sesuai dengan anotherNumber yang berada pada do. Maka tidak akan terjadi perulangan atau loop.

***Returns & Jumps***

Kotlin memiliki 3 jenis Jumps struktur expression seperti

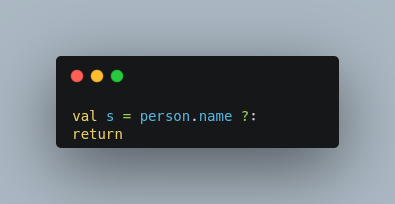
return -> Akan secara default me-return enclosing function terdekat atau anonymous function.

break -> Mematikan enclosing loop terdekat.

continue -> Melanjutkan proses pada enclosing loop terdekat

Semua expression diatas dapat dijadikan expression yang besar.

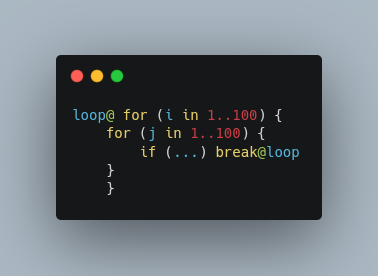
Contoh:

tipe yang digunakan adalah **Nothing Type**.

***Break & Continue Labels***

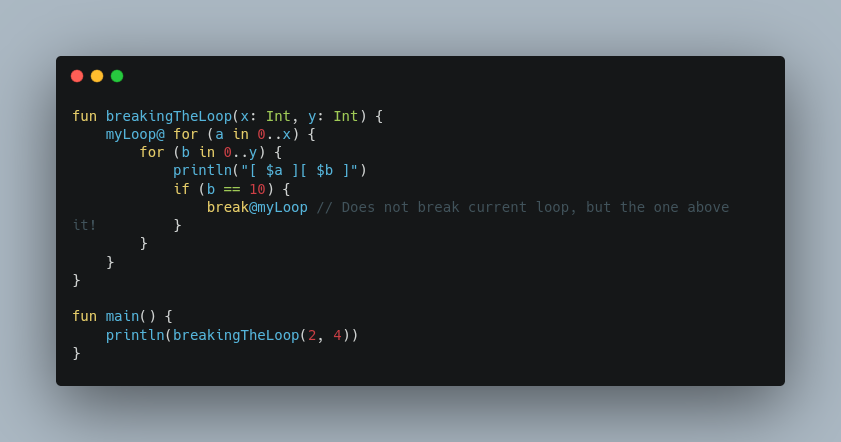
Semua expression pada Kotlin dapat di mark dengan sebuah label. Label mempunyai penanda yang diawali dengan @ seperti abc@ , anbidev@, fooBar@ adalah label yang valid (lihat grammar).

Untuk melabeli sebuah expression, kita hanya tinggal menginput label didepan sebuah expression.



Dan mengakhirinya dengan break@loop yang sama dengan label jumps dan dieksekusi tepat setelah loop ditandai dengan label.

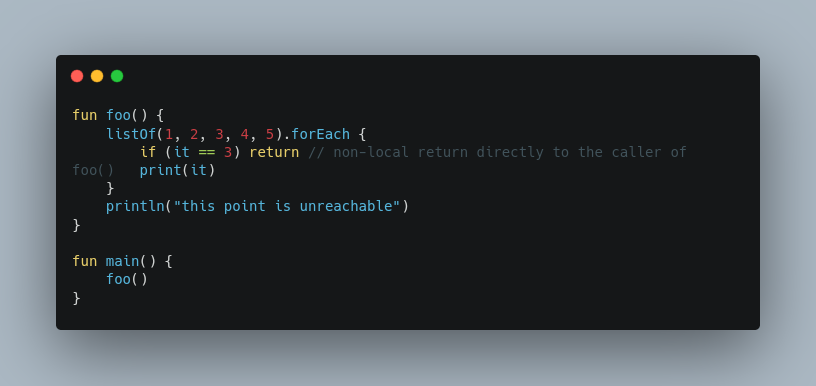
Contoh:



***Return at Labels up***

Dengan fungsi literals, local functions, object expressions dan juga functions dapat di nested dalam Kotlin. Fungsi returns yang terqualified memungkinkan Kita untuk return diluar function.

Dan yang terpenting adalah gunakan case untuk men-return dari sebuah lambda expression. Recall dengan kode seperti ini:



Return expressions men-return dengan enclosing function terdekat seperti foo.

PS : non-local return hanya dapat disupport dengan lambda expressions yang dipass menjadi inline function

Jika Kita perlu men-return sebuah lambda expression, Kita harus memberikan label dan mengkualifikasikannya pada return.



kode diatas hanya men-return dari lambda expression. Untuk selanjutnya, kita dapat membuatnya lebih mudah untuk menggunakan label yang implisit.

Seperti label yang memiliki nama yang sama dengan function dimana akan membuah lamba expression dapat lewat.

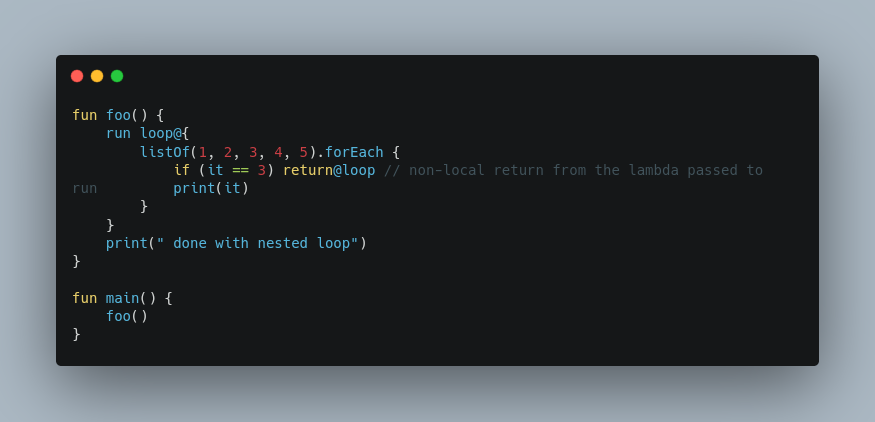


Kalian juga dapat menggantikan lambda expression dengan anonymous function. Statement return pada anonymous function akan men-return pada dirinya sendiri.



**Catatan :**

Menggunakan local function return seperti contoh diatas hampir sama dengan continue pada loop. Tidak ada tujuan yang digunakan untuk break, tapi kita dapat mensimulasikannya dengan menambahkan lambda dan juga non-locally return pada contoh :



Ketika men-return sebuah nilai, parse memberikan preference untuk qualified return seperti contoh:



yang berarti “return 1 pada label @a” dan bukan “mengembalikan label expression (@a 1)”.