



# Control Flow

**Silver**- Chapter 1 - Topic 3

---

Selamat datang di **Chapter 1 Topic 3** online  
course **Android Developer** dari Binar Academy!





## Haaii Binarian 🙋

### Masih di Chapter 1 nih~

Pada topik sebelumnya, kita udah belajar **dasar-dasar pemrograman Android**. Mulai dari struktur data, variabel, dan tipe data.

Pada topik kali ini kita akan mempelajari **Control Flow** sebagaimana Control Flow ini memang banyak digunakan pada bahasa pemrograman di Kotlin.

Apa sih Control Flow itu? Yuk kita cek~





## Detailnya, kita bakal bahas hal-hal berikut ini :

- Pengenalan Control Flow
- If-Else
- When
- Range
- For-Loop, While-Do, dan Do-While
- Return, Break, dan Continue
- Operators





Oke, sekarang kita akan berkenalan dulu dengan **Control Flow**.

Apa sih **Control Flow** itu?

Penjelasan lebih lengkap bisa lihat [disini~](#)





### Apa sih Control Flow itu?

**Control Flow** dalam bahasa pemrograman Kotlin bermanfaat untuk **melakukan perintah yang memiliki berbagai macam kondisi** dan **berulang**.

Memahami control flow ini super penting karena kalau kita belum memahami dasar control flow, kita belum bisa membuat sebuah logika berpikir yang benar dan juga efisien.





### Biar Gampang Memahami Control Flow, Bayangkan ini .....

Kamu kerja di sebuah restoran siap saji. Kalau bekerja di restoran terkenal, rasa dan kualitas adalah segalanya 😊

Ternyata setiap hari ada 1000 order yang masuk ke restoran. Selain menjamin kualitas, tim dapur perlu menyiapkan berbagai jenis masakan yang sama hingga 1000x.

Mungkinkah setiap makanan siap sajinya memiliki rasa yang sama, dan juga cepat di masaknya?





Untuk memastikan kualitasnya terjamin, **restoran akhirnya membuat resep dan flow koordinasi** dengan tim dapur.

Tujuannya supaya proses memasak bisa mengacu pada resep dan setiap orang di dapur tahu proses-proses yang perlu di lakukan

Nah, proses inilah yang disebut dengan **control flow**. Memastikan bahwa setiap pesanan memiliki kualitas dan rasa yang sama, bahkan setelah 1000x dibuat 😊







## Terus, di Pemograman Kotlin, Control flow itu yang kayak gimana?

Dalam pemograman kotlin, Control Flow akan dideklarasikan dalam beberapa sintaks, dibantu dengan beberapa ekspresi sintaks dibawah ini :

1. If - Else Expressions
2. When Expressions
3. Range
4. For - Loop
5. While-Do dan Do-While
6. Break dan Continue





Gengs bayangin kamu perlu mengunjungi rumah bestie satu-persatu untuk kasih nasi syukuran~

Nah orang pertama yang bakal kita kunjungi adalah **If-Else Expression** 🤪





### Kenalan dengan IF-Else Expression

Ketika membuat sebuah aplikasi, akan ada alur program yang mengarahkan kita untuk memutuskan sebuah kondisi (statement atau expression).

**If-else expression** digunakan untuk menjawab suatu pertanyaan kalau kondisinya memenuhi syarat yang bernilai **true (benar)**. Dan sebaliknya kalau bernilai **false (salah)** maka prosesnya akan dilewatkan.

Contohnya bisa dilihat pada kode disamping :

```
if (kondisi-1){  
    // eksekusi perintah 1  
} else {  
    // eksekusi perintah lainnya  
}
```



## Kenalan dengan IF-Else Expression

Pada contoh di samping, program akan mengecek apakah kondisi-1 memenuhi syarat yang ditentukan.

Jika sesuai dengan syarat yang dibuat kondisi-1, maka dia akan masuk ke dalam blok eksekusi perintah 1.

Jika tidak, maka dia akan masuk ke dalam blok eksekusi yang ada di else.

**Mudah kan, logikanya?**



```
if (kondisi-1){  
    // eksekusi perintah 1  
} else {  
    // eksekusi perintah lainnya  
}
```



### Kenalan dengan IF-Else Expression

Pada if-else expression sendiri memiliki beberapa tipe, diantaranya:

- If-else expression (if-else yang standar)
- If-else if-else expression (If-else dalam If-else)
- Nested if expression





Ada tiga saudara kembar **If-Else expression**, tapi apa sih keunikan dari mereka bertiga?





### IF-Else Expression yang standar

Dengan tipe if-else expression biasa, umumnya kita hanya akan membuat 1 kondisi saja.

Seperti contoh di samping, kita hanya membuat satu pengkondisian saja untuk menyaring sebuah kondisi.

Ketika kondisi pertama sesuai dengan yang ditentukan, maka blok else tidak akan tereksekusi.

```
val nilai = 50
if (nilai > 50){
    println("Lulus")
} else {
    println("Tidak Lulus")
}
```



### If-Else dalam If-Else

Pada **if-else if else ladder expression**, kita bisa membuat beberapa percabangan untuk cek suatu kondisi.

Seperti contoh di samping, kita bisa bikin beberapa pengkondisian layaknya sebuah tangga dan menyaring beberapa kondisi.

```
val nilai = 10
if (nilai > 0){
    println("Angka positif")
} else if (nilai < 0){
    println("Angka negatif")
} else {
    println("Angkanya 0")
}
```





### Nested If Expression

Nested if expression memungkinkan kita untuk menyaring sebuah kondisi, dan menyaringnya lagi jika kondisinya bernilai true.

Tipe ini cukup complex karena terjadi banyak penyaringan. Namun dalam proses memenuhi target kita, kompleksitas dalam membuat program perlu kita hadapi.

```
val nilai1 = 25
val nilai2 = 20
val nilai3 = 30
val hasil = if (nilai1 > nilai2) {
    val max = if (nilai1 > nilai3) {
        nilai1
    } else {
        nilai3
    }
    "body of if $max"
} else if (nilai2 > nilai3) {
    "body of else if $nilai2"
} else {
    "body of else $nilai3"
}
println(hasil)
```



Setelah kenalan sama **If-else bersaudara**,  
sekarang kita mampir ke rumah Ibu  
**When Expression**.

Apa sih **When Expression** itu?






### Kenalan dengan When Expression

Pada pemrograman Kotlin, **when** merupakan pengganti switch pada pemrograman Java.

Fungsi when hampir sama kayak **if-else** (memilih suatu proses yang akan dieksekusi dengan kondisi tertentu) bedanya penulisan syntax **when** lebih mudah dipahamin

Contoh sederhananya seperti gambar di samping



```
val number = 4
val hasil = when (number) {
    1 -> "Setunggal"
    2 -> "Kalih"
    3 -> "Tigo"
    else -> "Nomor tidak valid"
}

println(hasil)
```



### Kenalan dengan When Expression

Dengan menggunakan **when**, kita juga dapat memeriksa instance dengan tipe tertentu dari sebuah objek menggunakan **is** atau **!is**.

Contohnya seperti kode disamping.



```
val tipeData : Any = 10.0f

when(tipeData){
    is Float -> println("Float")
    is String -> println("Integer")
    else -> println("Else")
}
```



Tiga saudara kembar udah, Ibu temen kita udah. Sekarang kita akan mampir ke keluarga **Range Expression**.

Apa sih **Range Expression** itu?





## Kenalan dengan Range Expression

**Range** merupakan tipe data yang berisi urutan nilai dimana terdapat nilai awal dan nilai akhir. **Range** ini masuk ke golongan tipe data unik yang dibuat oleh Kotlin lho~

Kenapa unik? Kita coba lihat beberapa operator atau fungsi dari **Range** dibawah ini :

- **until**,
- **rangeTo()**, dan
- **downTo()**.





## Kenalan dengan Range Expression

```
val mRange = 0..10
```

Kode di atas menggunakan operator “..” untuk membuat range. Pada variable di atas berarti kita membuat sebuah variable yang berisikan nilai 0,1,2,3,4,5,6,7,8,9,10.

Kita bisa menambahkan jarak antara dua nilai dengan menambahkan step (secara default, step memiliki nilai 1) Nggak cuma itu, kita bisa mengubahnya sesuai kebutuhan kita, seperti contoh di bawah ini:

```
val mRange = 0..10 step 2
```

Kalau liat kode diatas, setelah kita menambahkan step dengan nilai 2, maka nilai yang akan dihasilkan menjadi 0,2,4,6,8,10





## Kalau Penggunaan rangeTo()

Selanjutnya kita akan menggunakan operator **rangeTo()** untuk membuat range. Contohnya seperti berikut :

```
val mRange = 0.rangeTo(10)
```

Pada kode di atas, nilai yang ditampilkan oleh variable tersebut sama dengan ketika kita membuatnya dengan manual menambahkan nilai 0 - 10.

Kita juga bisa menambahkan step seperti operator sebelumnya. Contohnya seperti berikut:

```
val mRange = 0.rangeTo(10) step 2
```







## Kalau Penggunaan `downTo()`

Kita juga dapat membuat range yang memiliki nilai terbalik menggunakan operator **`downTo()`**. Seperti berikut :

```
val mRange = 10.downTo(0)
```

Pada kode di atas, nilai yang ditampung oleh variable tersebut sama dengan ketika kita membuatnya dengan dua operator sebelumnya. Hanya saja urutan datanya akan terbalik menjadi 10,9,8,7,6,5,4,3,2,1,0.

Kita juga bisa menambahkan step seperti operator sebelumnya. Contohnya seperti berikut:

```
val mRange = 10.downTo(0) step 2
```



## Kalau Penggunaan until

Dan kita juga dapat membuat range operator **until** yang tidak memasukkan nilai akhir. Seperti berikut:

```
val mRange = 0 until 5
```

Pada kode di atas, nilai yang ditampung oleh variable tersebut akan menjadi 0,1,2,3,4. Dan untuk nilai 5 tidak akan masuk ke dalam variable tersebut.

Melalui operator tersebut, bisa disimpulkan bahwa kita membuat sebuah range yang nilainya dimulai 0 dan kurang dari 5.





## Kalau range dan when di kombinasi

Nah uniknya tipe data **range** ini kita bisa ajak collab sama tipe data lain kayak **when**.

Contohnya kita pingin menyaring sebuah data raport kayak yang ada di sebelah. Kalau diliat, kita menyaring data raport menggunakan **range** dan **when**.

**When** akan mengecek satu persatu setiap branchnya dan ketika terdapat kondisi **range** yang cocok, maka kondisi tersebut akan dieksekusi langsung.

Gimana collab-nya? Ciamik kan~

```
val nilai = 71
when(nilai) {
    in 0..50 -> println("Nilai kamu: F")
    in 51..60 -> println("Nilai kamu: E")
    in 61..70 -> println("Nilai kamu: D")
    in 71..80 -> println("Nilai kamu: C")
    in 81..90 -> println("Nilai kamu: B")
    in 91..100 -> println("Nilai kamu: A")
    else -> println("Masuk remedial")
}
```



Sekarang kita lanjutin petualangan kita membawa nasi syukuran ke orang selanjutnya. Di depan udah ada **For Loop**.

Yuk kita ketuk pintunya, apa sih **For Loop** itu?





## Kenalan dengan Loop

**Loop** atau dalam bahasa Indonesia disebut **putaran/perulangan** adalah **kondisi ketika kita akan menampilkan data yang banyak berulang kali.**

Bayangin ketika kita perlu mencetak kalimat 100x pada program. Bisa aja kita deklarasikan sampai 100x, tapi nanti jari kita yang jadi keriting 😅

Ini lah perlunya **loop** agar kerjaan kita jadi ringan. **Loop** dalam pemrograman digunakan untuk **mengulang blok kode sampai kondisi tertentu terpenuhi.**



## Apa saja jenis-jenis Loop?

Nah, untuk meminimalisir keritingnya jari kita, ada beberapa jenis ekspresi **Loop** yang bisa kita pakai, yaitu :

- **for**,
- **while-do**, dan
- **do-while**.





## Kenalan dengan For Loop

**For loop** merupakan perulangan yang paling umum digunakan pada Kotlin. For loop menyediakan fleksibilitas untuk melakukan iterasi melalui segala jenis struktur data.

For loop dapat digunakan pada **Ranges**, **Collections**, **Arrays** dan iterator lainnya. Contohnya kode di samping :

Kode di atas merupakan contoh ketika melakukan perulangan menggunakan **Ranges** seperti di topik sebelumnya.



```
for (i in 1..3) {  
    println("Nilai: $i")  
}
```



## Cara Deklarasi For Loop?

Kode di samping dapat diartikan bahwa kita menginisialisasi sebuah variabel bernama "i" yang melakukan perulangan dari angka 1 sampai 3.

Jika kode dijalankan, maka hasil yang muncul akan jadi seperti gambar dibawah :

```
Nilai: 1  
Nilai: 2  
Nilai: 3
```

```
for (i in 1..3) {  
    println("Nilai: $i")  
}
```





## For-Loop untuk data Array atau List

Selain pada **ranges**, kita juga dapat menggunakannya untuk menampilkan data pada **array** ataupun **list**.

Pada kode di atas, perulangan akan mengulang untuk mencetak elemen yang terdapat pada variable **mList**.

```
val mList = listOf("Halo ", "Saya Akan ", "Menjadi Android Developer")  
  
for (i in mList) {  
    print(i)  
}
```





## For-Loop dengan indices atau withIndex

Kita juga bisa menggunakan **indices** atau **withIndex**, yaitu library yang tersedia di Kotlin untuk mendapatkan **index** maupun **value** di dalam **array** atau **list**.

Contohnya kayak gini nih :

```
for (i in mList.indices) {  
    print(i)  
}
```

```
for ((index, values) in mList.withIndex()) {  
    print("index: $index, value: $values")  
}
```





## Kenalan dengan For-Loop

Satu lagi nih 🤖 Kita juga bisa manfaatin salah satu ekstensi pada Kotlin yaitu **forEach** untuk menampilkan data dalam array atau list **tanpa harus membuat sebuah kondisi**.

Hal tersebut dikarenakan **forEach** akan melakukan perulangan hingga data pada sebuah array atau list habis.

Contohnya seperti berikut:

```
val mList = listOf("Halo ", "Saya Akan ", "Menjadi Android Developer")

mList.forEach {
    print(it)
}
```





Gimana nih menggunakan For-Loop menurut kalian?

Tapii, Nasi syukurannya tinggal sedikit lagi. Masih perlu kita anter~

Di sebrang jalan akan ada rumah upin ipin-nya pemograman yaitu **While Do** dan **Do While Loop!**





### Kenalan dengan While-Do

Untuk menggunakan `while-do` pada Kotlin, kita hanya perlu memasukkan keyword `while` dilanjutkan dengan kondisi dan diakhiri dengan blok body dari `while` itu sendiri.

While memiliki sifat **Entry Controlled Loop**, dimana, kondisi yang diberikan akan dievaluasi terlebih dahulu.

Maksudnya gini :

- Jika kondisi tersebut terpenuhi atau bernilai true, maka proses perulangan akan dijalankan.
- Jika kondisi yang diberikan tidak terpenuhi sedari awal, maka proses perulangan tidak akan dijalankan sama sekali.

```
var nilai = 1
while (nilai <= 5) {
    println("While loop $nilai")
    nilai++
}
```



### Cara deklarasi While-Do?

Contohnya, bisa dilihat di kode disamping, dimana menunjukkan bahwa kita membuat sebuah variabel bernama **nilai** dengan nilai **1**.

Di bawahnya dibuat kondisi dimana jika **variable nilai** bernilai  $\leq 5$ , maka program akan mengarahkan kita ke dalam blok kode **while** dan setelahnya variable counter akan ditambah 1.

Dikarenakan variable counter selalu bertambah 1 ketika masuk perulangan tersebut, perulangan **while** akan berhenti ketika variable **nilai** lebih dari 5.

```
var nilai = 1
while (nilai <= 5) {
    println("While loop $nilai")
    nilai++
}
```



## Kenalan lagi dengan Do-While

**While** punya saudara loh, namanya **do-while** 😊

Untuk memanggil fungsi tersebut, kita hanya memerlukan keyword **do**, lalu diikuti dengan blok kode dan diakhir dengan fungsi **while** dan kondisi. Contoh:

Berbedanya dengan **while**, **do-while** bersifat **Exit Controlled Loop**, di mana proses perulangan akan langsung dijalankan di awal :

- Jika telah selesai, barulah kondisi yang diberikan akan dievaluasi.

```
var nilai = 1
do {
    println("Do While loop $nilai")
    nilai++
} while (nilai <= 5)
```



### Cara deklarasi Do-While?

Contohnya bisa cek kode disamping. Dimana dieksekusi perulangan dalam blok kode **do** dan menambahkan value **1** pada **variabel nilai**.

Setelahnya barulah kita melakukan pengecekan menggunakan fungsi while apakah **variabel nilai** masih bernilai  $\leq 5$  atau tidak.

Jika benar (true) maka perulangan akan dilakukan lagi hingga kondisi berakhir.

```
var nilai = 1
do {
    println("Do While loop $nilai")
    nilai++
} while (nilai <= 5)
```



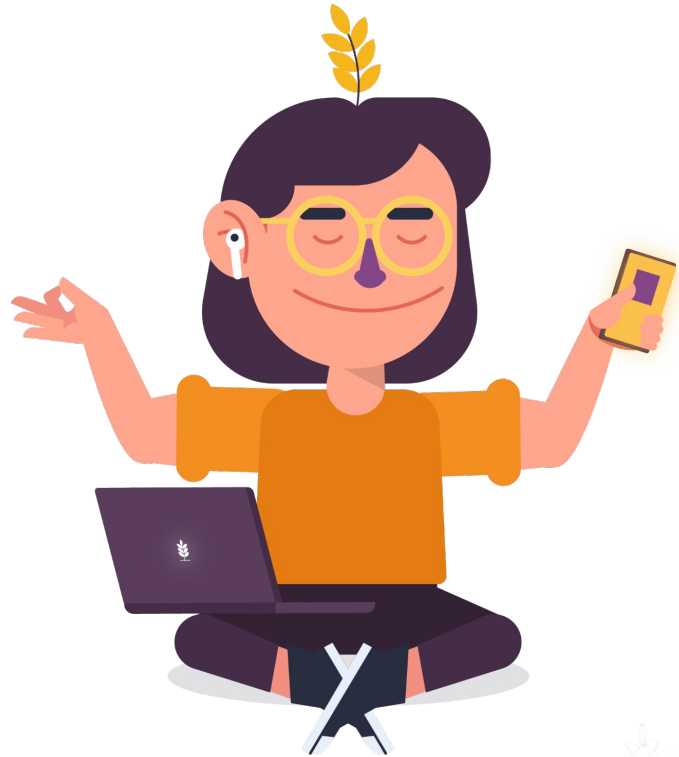


### Catatan penggunaan While-do dan Do-While

Pada saat menggunakan perulangan while-do dan do-while, kita harus memperhatikan agar terhindar dari **infinite loop** atau **perulangan tanpa akhir**.

Ketika hal tersebut terjadi, akan mengakibatkan proses perulangan menjalankan tugasnya tanpa henti.

Akibatnya, kode kamu bisa terus berjalan, sampai-sampai komputer kamu meleduk 🤖🔥





Perhentian selanjutnya akan berkenalan dengan Struktur **Jump Expression: Return, Break, dan Continue.**

Apa sih mereka itu? Lebih lengkapnya bisa cek penjelasan [disini~](#)





### Apa sih Jump Expression itu?

Ketika kita melakukan sebuah **perulangan / Loop**, terkadang kita dihadapkan dengan data yang tidak sesuai harapan atau sudah sesuai dengan yang kita harapkan.

Nah, dengan adanya **jump expression**, kita bisa berhenti ataupun melanjutkan eksekusi syntax.





### Apa sih Jump Expression itu?

Kotlin memiliki tiga **struktural jump expressions** yaitu **break** dan **continue**.

- Return -> Secara default kembali dari penutup function terdekat atau anonymous function.
- Break -> Mematikan enclosing loop terdekat.
- Continue -> Melanjutkan proses pada enclosing loop terdekat.

Selanjutnya kita pelajari satu persatu contohnya di slide berikut 🔥





### Cobain Break yuk ...

Pada kode disamping, dibuat sebuah variabel bernama **i** dengan **nilai 0** dan melakukan perulangan **while**. Dikondisikan **jika i < 10, maka akan mencetak i dan menambahkan variable i tersebut dengan nilai 1.**

Dalam perulangan terdapat pengecekan dimana jika **variabel i** bernilai sama dengan 4 maka akan melakukan **break**.

Saat itulah proses perulangan akan langsung berhenti dan menjalankan syntax di bawahnya.

```
var i = 0
while (i < 10) {
  println(i)
  i++
  if (i == 4) {
    break
  }
}
print("Perulangan akan berhenti di nilai: $i")
```



### Cobain Continue yuk ...

Pada kode di atas berarti kita melakukan perulangan dengan kondisi variabel *i* bernilai 0 hingga variabel *i* bernilai > 7. **Jika variabel *i* = 4 maka variable *i* akan ditambahkan 1.**

Dan ketika kita menjalankan syntax di atas, kita akan mengetahui bahwa nilai 4 akan terlongkap.

Hal tersebut dikarenakan variable *i* masuk ke dalam pengecekan kondisi pada blok **if** tetapi tidak diberhentikan melainkan dilanjutkan karena kita menggunakan fungsi **continue** di dalamnya.

```
var i = 0
while (i < 7) {
  if (i == 4) {
    i++
    continue
  }
  println(i)
  i++
}
```



## Untuk Return, kenalan dulu yuk ...

**Return** digunakan untuk mengembalikan sebuah value dari dalam function body. Kita dapat menganggap function sebagai mesin, lalu output dari mesin ini dikumpulkan oleh return statement untuk digunakan kembali.

Melalui kode disamping, function **nama()** mengembalikan string yang memiliki nilai "Binarian" dan nilai ini akan disimpan di dalam variable **myName**.

Jadi setiap kali function **nama()** dipanggil maka akan me-**return** atau mengembalikan nilai "**Binarian**".



```
fun main() {  
    val myName = name()  
    println("Name = $myName")  
}  
  
private fun name(): String {  
    return "Binarian"  
}
```



## Cara lain deklarasi Return ...

**Return** juga dapat digunakan dengan menggunakan pemanggilan berlabel dimana kontrol program nantinya akan mengembalikan ke baris tertentu dalam kode.

Misal, jika panjang kode kita adalah 100 baris dan kita memasukkan syntax **return** pada baris ke 45. Maka sisa kode berikutnya tidak akan tereksekusi.

```
fun foo2() {  
    listOf(1, 2, 3, 4, 5).forEach lit@{  
        if (it == 3) return@lit  
        // local return ke pemanggil lambda, yaitu forEach loop  
        print(it)  
    }  
    print(" selesai explicit label")  
}
```

```
fun foo1() {  
    listOf(1, 2, 3, 4, 5).forEach(fun(value: Int) {  
        if (value == 3) return  
        // local return ke pemanggil function anonymous, yaitu forEach loop  
        print(value)  
    })  
    print(" done with anonymous function")  
}
```

Selain itu kita bisa mengembalikan beberapa nilai dari **fungsi anonim** (fungsi yang tidak memiliki nama apapun) **Return** statement dalam fungsi anonim akan kembali dari **fungsi anonim** itu sendiri.





Rumah terakhir yang akan kita kunjungi adalah **Operators**.

Apa sih Operator? Penjelasan lebih lanjut bisa kamu intip [disini~](#)





## Apa sih Operator itu?

Kotlin memiliki berbagai macam **operator** yang dapat digunakan untuk melakukan proses aritmatika, perbandingan, dan lainnya.

**Operators** menggunakan simbol(karakter) khusus untuk melakukan operasi pada operan (variable dan value). Misalnya, “+” adalah operator yang melakukan penjumlahan.

Nah kamu nggak asing kan dengan simbol itu? kita akan pelajari peran beberapa simbol ketika digabungkan dengan bahasa Kotlin. Yuk mari~





## Macam-macam Operator

Pada Kotlin, operator dibagi menjadi:

- Arithmetic Operators (Operator Aritmatika)
- Assignment Operators (Operator Penugasan)
- Comparison Operators (Operator Perbandingan)
- Logical Operators (Operator Logika)

Seperti apa contohnya? Yuk kita pelajari di slide setelah ini~





## 1) Arithmetic Operators (Operator Aritmatika)

Seperti namanya operator aritmatika bertugas untuk melakukan berbagai tugas aritmatika. Seperti penjumlahan, pengurangan, dst.

Beberapa jenis operator aritmatika:

- (+) -> untuk melakukan penjumlahan atau bisa juga digunakan untuk menggabungkan *string*.
- (-) -> untuk melakukan pengurangan.
- (\*) -> untuk melakukan perkalian.
- (/) -> untuk melakukan pembagian.
- (%) -> untuk melakukan modulus atau sisa dari pembagian.

Operator	Function	Name	Description
+	plus()	Addition	Adds together two values
-	minus()	Subtraction	Subtracts one value from another
*	times()	Multiplication	Multiplies two values
/	div()	Division	Divides one value by another
%	rem()	Modulus	Returns the division remainder
++	inc()	Increment	Increases the value by 1
--	dec()	Decrement	Decreases the value by 1



## Contoh :

Untuk contoh penggunaannya, kita dapat melihat kode di samping.

```
fun main() {  
    val nomor1 = 12.5  
    val nomor2 = 3.5  
    var hasil: Double  
  
    hasil = nomor1 + nomor2  
    println("nomor1 + nomor2 = $hasil") // hasil: 16.0  
  
    hasil = nomor1 - nomor2  
    println("nomor1 - nomor2 = $hasil") // hasil: 9.0  
  
    hasil = nomor1 * nomor2  
    println("nomor1 * nomor2 = $hasil") // hasil: 43.75  
  
    hasil = nomor1 / nomor2  
    println("nomor1 / nomor2 = $hasil") // hasil: 3.5714285714285716  
  
    hasil = nomor1 % nomor2  
    println("nomor1 % nomor2 = $hasil") // hasil: 2.0  
}
```



Enggak cuma itu. ternyata simbol “+” juga dapat digunakan untuk menyambungkan beberapa kalimat yang terpecah menjadi satu kalimat?

Gak percaya? Bisa cek koding berikut :

```
fun main() {  
    val kAwal = "Ngomong itu mudah."  
    val kTengah = "Tunjukkan kodemu! "  
    val kAkhir = "- Binarian"  
  
    val hasil = kAwal + kTengah + kAkhir  
    println(hasil)  
    // hasil: Ngomong itu mudah. Tunjukkan kodemu! - Binarian  
}
```



## Arithmetic Operators - Operator Overloading

Kotlin juga membuat syntax aritmatika operator lain yang memudahkan kita untuk memahami cara bekerja suatu operator aritmatika yang disebut **operator overloading** (operator yang dapat menangani berbagai jenis tipe data)

Contoh Deklarasi  
Operator Overriding

```
// operator "+" untuk tipe basic
operator fun plus(other: Byte): Int
operator fun plus(other: Short): Int
operator fun plus(other: Int): Int
operator fun plus(other: Long): Long
operator fun plus(other: Float): Float
operator fun plus(other: Double): Double

// untuk penggabungan String
operator fun String?.plus(other: Any?): String
```



Di contoh tersebut, ketika kita menggunakan operator yang telah kita pelajari sebelumnya. Kotlin akan menyarankan kita untuk menggunakan **operator overloading** yang bertujuan agar operator tersebut dapat handle beberapa **tipe data** sekaligus.

Misalnya, kita menggunakan syntax "**a+b**", maka Kotlin akan menyarankan untuk merubahnya menjadi "a.plus(b)".

Operator "plus" dapat bekerja pada beberapa tipe data sekaligus (kecuali Char dan Boolean).

```
// operator "+" untuk tipe basic
operator fun plus(other: Byte): Int
operator fun plus(other: Short): Int
operator fun plus(other: Int): Int
operator fun plus(other: Long): Long
operator fun plus(other: Float): Float
operator fun plus(other: Double): Double

// untuk penggabungan String
operator fun String?.plus(other: Any?): String
```





## So, yang disebut Operator Overloading apa aja yaa?

Berikut adalah table dari **operator aritmatika** dan juga ketika kita menggunakan **operator overloading**

Ekspresi	Nama fungsi	Diartikan dengan
$a + b$	plus	<code>a.plus(b)</code>
$a - b$	minus	<code>a.minus(b)</code>
$a * b$	times	<code>a.times(b)</code>
$a / b$	div	<code>a.div(b)</code>
$a \% b$	mod	<code>a.mod(b)</code>



## 2) Assignment Operators (Operator Penugasan)

Operator penugasan digunakan untuk menambahkan nilai ke dalam sebuah variable. Mungkin kita sudah tidak asing, salah satu contoh dari operator penugasan adalah “=”.

Beberapa adalah tabel operator penugasan :

Ekspresi	Persamaan	Penjelasan
$a += b$	$a = a + b$	$a + b$ dan ditampung di variable $a$ kembali
$a -= b$	$a = a - b$	$a - b$ dan ditampung di variable $a$ kembali
$a *= b$	$a = a * b$	$a * b$ dan ditampung di variable $a$ kembali
$a /= b$	$a = a / b$	$a / b$ dan ditampung di variable $a$ kembali
$a \% = b$	$a = a \% b$	$a \% b$ dan ditampung di variable $a$ kembali



### Cara pakai Assignment Operators, gimana?

Bisa cek kode di samping, pertama-tama kita membuat sebuah variable dengan nama “**nomor**” dan dengan value **12**. Setelahnya kita melakukan **operator penugasan** dengan fungsi perkalian, dengan menggunakan syntax “**\*=**”.

Syntax di atas berarti melakukan proses variable nomor dengan value **12**, di kali dengan **5** dan hasilnya disimpan kembali ke dalam variable “**nomor**”.

Setelahnya kita menampilkan variable tersebut dengan menggunakan syntax **println** dan kita mendapatkan hasilnya yaitu **60**.



```
fun main() {  
    val nomor = 12  
  
    nomor *= 5  
    println("nomor1 * nomor2 = $hasil") // hasil: 60  
}
```



## Comparison Operators (Operator Perbandingan)

**Operator perbandingan** adalah operator yang digunakan untuk membandingkan 2 buah **operand**, hasil dari penggunaan operator ini adalah **boolean** (true or false).

Operator	Penjelasan	Ekspresi	Operator Overloading
>	lebih besar dari	$a > b$	<code>a.compareTo(b) &gt; 0</code>
<	kurang dari	$a < b$	<code>a.compareTo(b) &lt; 0</code>
>=	Lebih dari atau sama dengan	$a \geq b$	<code>a.compareTo(b) &gt;= 0</code>
<=	Kurang dari atau sama dengan	$a \leq b$	<code>a.compareTo(b) &lt;= 0</code>
==	Setara dengan	$a == b$	<code>a?.equals(b) ?: (b === null)</code>
!=	Tidak setara dengan	$a != b$	<code>!(a?.equals(b) ?: (b === null))</code>



### Cara pakai Operator Perbandingan?

**Operator perbandingan** biasanya digunakan bersamaan dengan control flow seperti **if-else**, **when** dan juga **looping**. Contoh penerapan operator perbandingan dapat kita lihat seperti kode disamping :

Penggunaan Operator Perbandingan lebih besar dari ">"

```
fun main() {  
    val a = -12  
    val b = 12  
  
    // menggunakan operator lebih besar dari  
    val hasil = if (a > b) {  
        println("a lebih besar dari b.")  
    } else {  
        println("b lebih besar dari a.")  
    }  
  
    println("hasil = $hasil")  
  
    /* OUTPUT  
    * b lebih besar dari a.  
    * hasil = 12  
    * */  
}
```



## 3) Logical Operators (Operator Logika)

**Operator logika** merupakan sebuah operator yang digunakan untuk menentukan logic diantara dua variable atau value. Operator logika sama dengan **operator pembandingan**, yaitu mengembalikan nilai dalam bentuk Boolean (true or false).

Operator	Nama	Penjelasan	Contoh
&&	Logical AND (Dan)	Mengembalikan nilai <b>true</b> jika kedua statement adalah benar.	$x < 5 \ \&\& \ x < 10$
	Logical OR (Atau)	Mengembalikan nilai <b>true</b> jika salah satu statement adalah benar.	$x < 5 \    \ x < 4$
!	Logical NOT (Negasi)	Mengembalikan hasil, mengembalikan nilai <b>false</b> jika hasilnya adalah <b>true</b> .	!x



```
fun main() {  
    val a = 10  
    val b = 5  
    val c = 15  
    val flag = false  
    var result: Boolean  
    result = (a > b) && (a > c)  
    println("(a>b) && (a>c) :$result")  
  
    result = (a > b) || (a > c)  
    println("(a>b) || (a>c) :$result")  
  
    result = !flag  
    println("!flag :$result")  
}  
  
/* OUTPUT  
* (a>b) && (a>c) :false  
* (a>b) || (a>c) :true  
* !flag :true  
* */
```

### Contoh :

Coba deh lihat kode di samping, kalau kita menjalankan operator logika, hasil yang didapat akan selalu true or false, dan jika kita menggunakan "!" maka akan membalikkan hasil yang di dapat.

Saatnya kita  
**Quiz!**







## 1. Dibawah ini, yang merupakan pernyataan benar terkait fungsi Break adalah?

- A. Break digunakan untuk keluar dari program Kotlin
- B. Break digunakan untuk men-debug program Kotlin
- C. Break digunakan untuk keluar dari loop



## 1. Dibawah ini, yang merupakan pernyataan benar terkait fungsi Break adalah?

- A. Break digunakan untuk keluar dari program Kotlin
- B. Break digunakan untuk men-debug program Kotlin
- C. Break digunakan untuk keluar dari loop

Seperti pembahasan yang sebelumnya telah kita pelajari bahwa benar break digunakan untuk keluar dari perulangan. Hal tersebut tentu sesuai dengan kondisi yang kita inginkan.



2. Perhatikan syntax dibawah ini.  
Berapakah angka terakhir yang akan dicetak oleh for loop?

```
fun main() {  
    for (item in 6 downTo 1 step 2) {  
        println(item)  
    }  
}
```

A. 6

B. 4

C. 2



**2. Perhatikan syntax dibawah ini.  
Berapakah angka terakhir yang akan dicetak oleh for loop?**

```
fun main() {  
    for (item in 6 downTo 1 step 2) {  
        println(item)  
    }  
}
```

A. 6

B. 4

C. 2

Ya, tepat sekali!

Ketika kita mengeksekusi kode tersebut, itu berarti kita melakukan perulangan secara terbalik dari 6 ke 1 dengan jarak 2. Jadi angka terakhir yang akan tampil pada perulangan tersebut adalah 2.



**3. Manakah dari syntax berikut ini yang tidak didukung oleh bahasa pemrograman Kotlin?**

- A. If...else if...else
- B. if...else
- C. if...then...else



**3. Manakah dari syntax berikut ini yang tidak didukung oleh bahasa pemrograman Kotlin?**

- A. **if-else if-else**
- B. **if-else**
- C. **if-then-else**

Betul!!!

Kotlin mendukung **if-else** expression, juga fungsi **if-else** dalam **if-else**  
Tapi, Kotlin sama sekali tidak mendukung syntax **if-then-else**



4. Perhatikan syntax berikut, berapakah angka terakhir yang akan dicetak oleh for loop?

```
var x = 20
var y = 15
var z = "Mangga"

val result = if (x > y) {
    z = "Jeruk"
} else {
    z = "Apel"
}
println("Hasilnya adalah = $z")
```

- A. Jeruk
- B. Mangga
- C. Apel



#### 4. Perhatikan syntax berikut, berapakah angka terakhir yang akan dicetak oleh for loop?

```
var x = 20
var y = 15
var z = "Mangga"

val result = if (x > y) {
    z = "Jeruk"
} else {
    z = "Apel"
}
println("Hasilnya adalah = $z")
```

- A. Jeruk
- B. Mangga
- C. Apel

Tepat sekali!!!

Jawaban yang benar adalah **Jeruk**.

Itu dikarenakan **x lebih besar dari y**, jadi ketika kondisi tersebut benar, variable z akan diberi nilai Jeruk dan bukannya Apel





## 5. Apakah perbedaan dari while-do dan do-while loop?

- A. While-do lebih cepat dari do-while
- B. Do-while lebih cepat dari while-do
- C. While-do mengecek kondisi terlebih dahulu kondisi dan setelahnya menjalankan proses, kebalikannya dengan do-while



## 5. Apakah perbedaan dari while-do dan do-while loop?

- A. While-do lebih cepat dari do-while
- B. Do-while lebih cepat dari while-do
- C. While-do mengecek kondisi terlebih dahulu kondisi dan setelahnya menjalankan proses, kebalikannya dengan do-while

Betul!!!

**While-do** akan mengecek terlebih dahulu kondisi yang tersedia baru setelahnya menjalankan proses hingga kondisi yang ditentukan bernilai false.

Sedangkan **do-while** akan menjalankan perintah terlebih dahulu, barulah di akhir akan mengecek kondisinya.



Nah, selesai sudah pembahasan kita di Chapter 1 Topic 3 ini.

Selanjutnya, kita bakal bahas **Class and Object**.

Penasaran kayak gimana? Cus langsung ke topik selanjutnya~



Terima Kasih!



Next Topic

loading...