



## Kotlin

**XML, Gradle, and Principles of Kotlin**

## Daftar Isi

<b>A. ViewBinding</b>	<b>3</b>
<b>B. Gradle</b>	<b>5</b>
<b>C. Kotlin</b>	<b>9</b>
<b>D. Tipe Data pada Kotlin</b>	<b>10</b>
<b>E. Variabel pada Kotlin</b>	<b>11</b>
<b>F. Percabangan</b>	<b>14</b>
<b>G. Perulangan</b>	<b>19</b>
<b>H. Function</b>	<b>22</b>
<b>References</b>	<b>30</b>

## A. ViewBinding

View binding adalah fitur yang memudahkan kita menulis kode yang berinteraksi dengan tampilan (UI). Setelah diaktifkan dalam sebuah modul, view binding akan menghasilkan class binding untuk setiap file tata letak XML yang ada dalam modul tersebut. Instance class binding berisi referensi langsung ke semua tampilan yang memiliki ID di tata letak yang terkait. Pada umumnya, view binding menggantikan findViewById.

View binding diaktifkan di level modul. Untuk mengaktifkan view binding dalam modul, tambahkan elemen viewBinding ke file build.gradle-nya, seperti yang ditunjukkan pada contoh berikut:

```
android {  
    ...  
    viewBinding {  
        enabled = true  
    }  
}
```

Jika view binding diaktifkan untuk sebuah modul, sebuah class binding akan dihasilkan untuk setiap file tata letak XML yang berada di dalam modul. Setiap class binding berisi referensi ke tampilan root dan semua tampilan yang memiliki ID. Nama class binding dihasilkan dengan mengonversi nama file XML menjadi camel case dan menambahkan kata "Binding" ke bagian akhirnya. Misalnya, pada file tata letak dengan nama result\_profile.xml :

```
<LinearLayout ... >
    <TextView android:id="@+id/name" />
    <ImageView android:cropToPadding="true" />
    <Button android:id="@+id/button"
        android:background="@drawable/rounded_button" />
</LinearLayout>
```

Class binding yang dihasilkan dari layout XML di atas nantinya disebut `ResultProfileBinding`. Class ini memiliki dua kolom: `TextView` yang disebut `name` dan `Button` yang disebut `button`. `ImageView` dalam file tata letak di atas tidak memiliki ID, sehingga tidak ada referensi class tersebut dalam class binding ini.

Setiap class binding juga mencakup metode `getRoot()`, yang menyediakan referensi langsung untuk tampilan root file tata letak yang terkait. Dalam contoh ini, metode `getRoot()` dalam class `ResultProfileBinding` akan menampilkan tampilan root `LinearLayout`.

## Menggunakan View Binding pada Activity

Untuk menyiapkan instance class binding agar dapat digunakan dengan suatu activity, lakukan langkah-langkah berikut dalam metode `onCreate()` activity :

1. Panggil metode `inflate()` statis yang disertakan dalam class binding yang dihasilkan. Tindakan ini membuat instance dari class binding yang akan digunakan activity.
2. Dapatkan referensi ke tampilan root dengan memanggil metode `getRoot()` atau menggunakan sintaksis properti Kotlin.
3. Teruskan tampilan root ke `setContentView()` untuk menjadikan tampilan root tersebut sebagai tampilan aktif di layar.

```
private lateinit var binding: ResultProfileBinding

override fun onCreate(savedInstanceState: Bundle) {
    super.onCreate(savedInstanceState)
    binding = ResultProfileBinding.inflate(layoutInflater)
    val view = binding.root
    setContentView(view)
}
```

Contoh implementasi menyiapkan view binding pada activity

```
binding.name.text = viewModel.name
binding.button.setOnClickListener { viewModel.userClicked() }
```

Contoh implementasi penggunaan instance class binding

## B. Gradle

Sistem build Android mengkompilasi resource dan source code aplikasi lalu memaketkannya menjadi APK atau Android App Bundle (AAB) yang dapat kita uji, deploy, sign, dan distribusikan. Android Studio menggunakan Gradle, sebuah toolkit build canggih, untuk mengotomatiskan dan mengelola proses build, sekaligus memungkinkan kita menentukan konfigurasi build kustom yang fleksibel. Setiap konfigurasi build dapat menentukan rangkaian kode dan resource-nya sendiri, sekaligus menggunakan kembali bagian-bagian yang ada di semua versi aplikasi yang dikembangkan. Plugin Android untuk Gradle berfungsi dengan toolkit build ini untuk menyediakan proses dan setelan yang dapat dikonfigurasi khusus untuk mem-build dan menguji aplikasi Android. Pada dasarnya, sebuah proyek baru di Android Studio hanya memiliki 1 (satu) modul, yaitu app. Dengan begitu, untuk mengatur

dependensi pun cukup mudah karena terpusat pada sebuah berkas build.gradle (app).

## Glosarium build Android

Gradle dan plugin Android Gradle membantu developer Android mengonfigurasi aspek-aspek build berikut :

### 1. Build types (Jenis build)

Build types menentukan properti tertentu yang digunakan Gradle ketika mem-build dan memaketkan aplikasi. Build types biasanya dikonfigurasi untuk berbagai tahap siklus proses pengembangan.

Misalnya, build types debug mengaktifkan opsi debug dan menandatangani aplikasi dengan kunci debug, sedangkan build types release dapat menyusutkan ukuran, meng-obfuscate, dan menandatangani aplikasi dengan kunci release untuk distribusi. Kita harus menentukan setidaknya satu build types untuk mem-build aplikasi. Android Studio membuat build types release dan debug secara default.

### 2. Product flavors (Ragam produk)

Product flavors merepresentasikan berbagai versi aplikasi yang dapat dirilis kepada user, seperti versi gratis dan berbayar. Kita dapat menyesuaikan product flavors untuk menggunakan kode dan resource yang berbeda sekaligus berbagi dan menggunakan kembali bagian-bagian yang umum untuk semua versi aplikasi. Product flavors bersifat opsional, dan kita harus membuatnya secara manual.

### 3. Build variant (Varian build)

Build variant adalah cross product dari jenis build dan product flavors, dan merupakan konfigurasi yang digunakan Gradle untuk mem-build aplikasi. Dengan build variant, kita dapat mem-build versi debug product flavors selama pengembangan dan menandatangani versi rilis product flavors untuk distribusi. Meskipun tidak harus mengkonfigurasi build variant secara langsung, kita perlu mengkonfigurasi jenis build dan ragam produk yang membentuknya. Membuat jenis build atau ragam produk tambahan juga akan membuat varian build tambahan.

### 4. Manifest entries

Kita dapat menentukan nilai untuk beberapa properti file manifes dalam konfigurasi build variant. Nilai build ini menggantikan nilai yang ada dalam file manifest. Ini berguna jika kita ingin membuat beberapa varian aplikasi dengan nama aplikasi, versi SDK minimum, atau versi SDK target yang berbeda.

### 5. Dependencies

Sistem build mengelola dependensi project dari sistem file lokal dan dari repositori jarak jauh. Ini berarti kita tidak perlu menelusuri, mendownload, dan menyalin paket biner dependensi secara manual ke dalam direktori project.

Pengembangan sebuah aplikasi tak pernah lepas dari peran library. Baik itu library pihak ke-3 ataupun standard library. Pada proyek Android, penambahan sebuah library bisa dilakukan dengan cara

menambahkannya pada file build.gradle sebagai dependensi. Kita bisa mudah melakukannya dengan bantuan Gradle build system.

```
dependencies {  
    implementation(fileTree(mapOf("dir" to "libs", "include" to listOf("*.jar"))))  
    implementation("org.jetbrains.kotlin:kotlin-stdlib-jdk7:1.3.31")  
    implementation("com.android.support:appcompat-v7:28.0.0")  
    implementation("com.android.support.constraint:constraint-layout:1.1.3")  
    testImplementation("junit:junit:4.12")  
    androidTestImplementation("com.android.support.test:runner:1.0.2")  
    androidTestImplementation("com.android.support.test.espresso:espresso-core:3.0.2")  
}
```

Contoh penambahan library pada dependencies gradle

## 6. Signing (Penandatanganan)

Sistem build memungkinkan kita menentukan setelan signing dalam konfigurasi build dan dapat otomatis menandatangani aplikasi selama proses build. Sistem build menandatangani versi debug dengan sertifikat dan kunci default menggunakan kredensial yang dikenal untuk menghindari permintaan sandi pada waktu build. Sistem build tidak menandatangani versi release kecuali kita secara eksplisit menentukan konfigurasi penandatanganan untuk build ini. Build release yang ditandatangani diperlukan untuk mendistribusikan aplikasi melalui sebagian besar app store.

## 7. Code and resource shrinking (Penyingkatan ukuran kode dan resource)

Sistem build memungkinkan kita menentukan file aturan ProGuard yang berbeda untuk setiap build variant. Saat mem-build aplikasi, sistem build akan menerapkan rangkaian aturan yang sesuai untuk menyingkat kode dan resource menggunakan alat penyingkat bawaan, seperti R8.

Menyingkat kode dan resource dapat membantu mengurangi ukuran APK atau AAB.

## 8. Dukungan multi-APK

Sistem build memungkinkan kita untuk otomatis mem-build berbagai APK yang masing-masing hanya berisi kode dan resource yang dibutuhkan untuk kepadatan layar tertentu atau Antarmuka Biner Aplikasi (ABI). Namun, merilis satu AAB adalah pendekatan yang direkomendasikan, karena memberikan pemisahan menurut bahasa selain kepadatan layar dan ABI, sekaligus tidak perlu mengupload beberapa artefak ke Google Play. Semua aplikasi baru yang dikirimkan setelah Agustus 2021 harus menggunakan AAB.

## C. Kotlin

Kotlin adalah bahasa pemrograman open-source yang dikembangkan oleh JetBrains untuk berbagai platform. Tapi, bahasa pemrograman ini semakin populer digunakan untuk membangun aplikasi Android.

Bahasa Kotlin berjalan pada platform Java Virtual Machine (JVM), sebuah platform yang memungkinkan komputer menjalankan kode berbasis Java, atau kode dari bahasa lain yang dikompilasi (compile) menggunakan Java. Artinya, Kotlin bisa menerapkan mekanisme compile pada Java. Bagi yang belum tahu, Java adalah bahasa yang bisa dipakai untuk menulis kode lalu menjalankannya di platform yang mendukung (Write Once Run Anywhere). Bahkan, Kotlin juga bisa digunakan bersama dengan Java. Dengan kata lain, kita bisa menggunakan Kotlin dan Java sekaligus untuk membangun satu

aplikasi Android. Contohnya, kita bisa menggunakan Kotlin untuk membuat satu halaman aplikasi, sementara halaman lainnya ditulis menggunakan Java. Kedua bahasa tersebut tidak akan bentrok satu sama lain karena menghasilkan output file yang sama.

Bahasa pemrograman Kotlin tergolong bahasa yang diketik secara statis (statically typed). Artinya, semua variabel yang dimasukkan ke dalam program harus dikenalkan terlebih dahulu apa jenisnya. Dengan begitu, error yang terjadi saat menulis kode (coding) dapat terdeteksi saat itu juga. Sehingga pada saat proses compile, program sudah benar-benar bersih dari error.

## D. Tipe Data pada Kotlin

Tipe data adalah jenis nilai yang akan disimpan oleh sebuah variabel. Ada beberapa jenis tipe data dalam Kotlin :

1. Int (Integer): bilangan bulat, contoh 12, 1\_000
2. String: teks, contoh "belajar kotlin asik"
3. Float: bilangan pecahan, contoh 21.23, 1.3
4. Double: bilangan pecahan juga, tapi punya ukuran penyimpanan yang lebih besar dari Float.
5. Boolean: tipe data yang hanya bernilai true dan false
6. Char: karakter, contoh 'A'
7. Unit: Tipe data yang hanya punya satu nilai, yaitu: objek Unit. Tipe data ini mirip seperti void pada Java.

Kita dapat mengonversi (mengubah) tipe data suatu variabel dengan beberapa metode yang sudah disediakan oleh Kotlin, di antaranya :

1. `toInt()` untuk mengubah ke integer
2. `toFloat()` untuk mengubah ke float
3. `toString()` untuk mengubah ke String
4. `toDouble()` untuk mengubah ke Double
5. `toLong()` untuk mengubah ke long integer
6. `toShort()` untuk mengubah ke short integer
7. `toChar()` untuk mengubah ke Char atau karakter

## E. Variabel pada Kotlin

Pembuatan variabel di Kotlin tidak terlalu formal seperti di Java. Pada Kotlin, kita boleh tidak menentukan/menyebutkan tipe datanya. Karena Kotlin sudah mampu mengenali tipe data dari nilai yang akan kita berikan. Pembuatan variabel diawali dengan kata kunci `var` dan `val`. Berikut ini merupakan contoh pembuatan variabel dengan tipe data pada Kotlin :

```
// membuat variabel kosong (Wajib menyebut tipe data)
var namaLengkap: String

// membuat variabel dan langsung diisi
// (tidak wajib menyebut tipe data, karena sudah punya nilai)
var alamat: String = "Mataram"
var tanggalLahir = "05-11-1993" as String
```

Berdasarkan sifatnya, variabel dalam kotlin dibagi menjadi dua jenis.

1. **Immutable:** read only
2. **Mutable:** read and write

Immutable artinya hanya sekali pakai, variabel ini seperti konstanta. Variabel immutable tidak bisa diisi ulang lagi nilainya alias read only. Pembuatan variabel immutable menggunakan kata kunci **val**. Contoh :

```
val tanggalLahir = "12-02-1995"
val jenisKelamin = "Pria"

// jika kita coba isi ulang nilainya, maka akan terjadi error
// karena variabel ini bersifat imutable
jenisKelamin = "Perempuan"
```

Sedangkan variabel mutable adalah variabel yang bisa diisi lagi nilainya. Pembuatan variabel mutable menggunakan kata kunci **var**. Contoh :

```
var jabatan = "Programmer"

// isi ulang nilainya
jabatan = "Project Manager"

println(jabatan) // output: Project Manager
```

Ada beberapa aturan penulisan variabel di Kotlin yang sebaiknya ditaati agar valid dan tidak error yaitu sebagai berikut :

1. Variabel kosong yang belum diberikan nilai wajib disebutkan tipe datanya.
2. Penulisan nama variabel menggunakan gaya camelCase.
3. Nama variabel tidak boleh diawali dengan angka dan simbol
4. Nama variabel tidak boleh menggunakan simbol, kecuali garis bawah atau underscore.
5. Tipe data diawali dengan huruf kapital

Setiap variabel dalam bahasa Kotlin mempunyai lingkupnya masing-masing. Variabel di lingkup yang sama tidak boleh punya nama yang sama. Saat variabel dideklarasikan di dalam function, variabel tersebut hanya bisa digunakan di dalamnya.

Jika deklarasi variabel dilakukan di luar function, maka variabel tersebut bisa digunakan di dalam file tempat deklarasinya dan di file yang menyertakannya. Sederhananya, ada dua jenis variabel berdasarkan tempat deklarasinya.

1. **Variabel global** : Variabel yang dideklarasikan di luar function. Variabel ini bisa digunakan semua function di dalam satu file. Deklarasinya sebaiknya di atas function main.
2. **Variabel lokal** : Variabel yang dideklarasikan di dalam compound statement dari function, loop, atau percabangan. Variabel tersebut hanya bisa digunakan di dalam function atau di antara tanda kurung kurawal ( {....} ) yang mengelilinginya.

Kita bisa mendeklarasikan variabel lokal dengan nama yang sama di beberapa function berbeda. Walaupun namanya sama, setiap variabel lokal di function yang berbeda dianggap sebagai beberapa variabel yang berbeda. Berikut ini merupakan contoh implementasi sederhana dari variabel global dan lokal pada kode Kotlin :

```
// Global variable
var fullName = "Siti Astuti"

fun main() {
    // Local variable
    var address = "Jakarta"

    print("$fullName tinggal di $address")
    // Output :
    // Siti Astuti tinggal di Jakarta
}
```

## F. Percabangan

Percabangan atau control Flow adalah sebuah istilah untuk menyebut alur program yang bercabang. Berikut ini merupakan bentuk percabangan yang terdapat pada Kotlin :

### 1. Percabangan if

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

Sintaks dasar percabangan if :

```
if (kondisi) {
    // blok kode yang akan dieksekusi di ini
}
```

Perhatikan bagian kondisi, di sana kita bisa isi dengan sebuah variabel boolean atau pernyataan yang menghasilkan nilai boolean.

Contoh kode program percabangan if :

```
fun main(args: Array<String>){

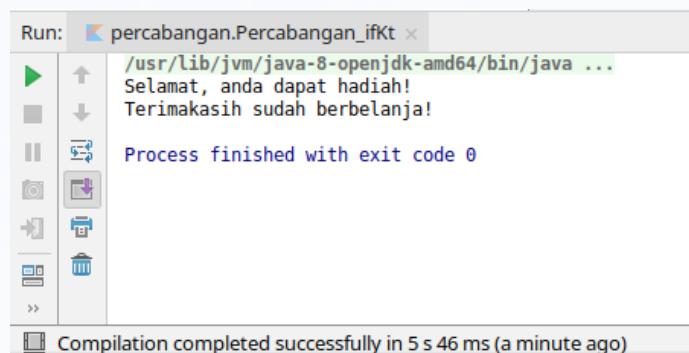
    val totalBelanja: Int = 192_000

    if (totalBelanja >= 100_000){
        println("Selamat, anda dapat hadiah!")
    }

    println("Terimakasih sudah berbelanja!")

}
```

Output dari kode program di atas :



The screenshot shows a Java application window titled "percabangan.Percabangan\_ifKt". The "Run" tab is selected. The output pane displays the following text:  
"/usr/lib/jvm/java-8-openjdk-amd64/bin/java ...  
Selamat, anda dapat hadiah!  
Terimakasih sudah berbelanja!  
Process finished with exit code 0  
Compilation completed successfully in 5 s 46 ms (a minute ago)

## 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.

Sintaks dasar percabangan if/else :

```
if (kondisi) {
    // blok kode yang akan dieksekusi
    // jika kondisi bernilai true
} else {
    // blok kode yang akan dieksekusi
    // jika kondisi bernilai false
}
```

Contoh kode program percabangan if/else :

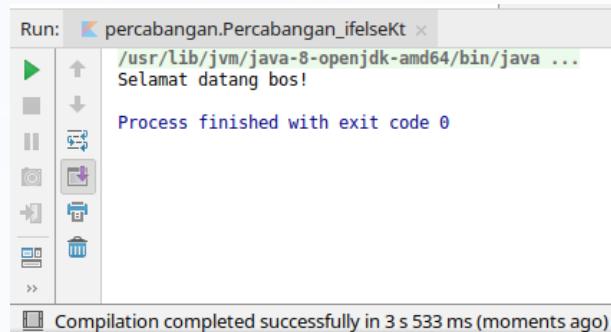
```
fun main(args: Array<String>){

    val password: String = "kopi"

    if (password == "kopi"){
        println("Selamat datang bos!")
    } else {
        println("Siapa kamu? pergi sana!")
    }

}
```

Output dari kode program di atas :



The screenshot shows a Java application window titled 'percabangan.Percabangan\_ifelseKt'. The 'Run' tab is selected. The output pane displays the following text:  
/usr/lib/jvm/java-8-openjdk-amd64/bin/java ...  
Selamat datang bos!  
Process finished with exit code 0  
  
At the bottom, a message indicates: Compilation completed successfully in 3 s 533 ms (moments ago).

### 3. Percabangan if/else if/else

Percabangan ini memiliki lebih dari dua blok kode dan kondisi.

Sintaks dasar percabangan if/else if/else :

```
if(kondisiPertama){
    // blok kode yang akan dieksekusi
    // jika kondisiPertama bernilai true
} else if(kondisiKedua) {
    // blok kode yang akan dieksekusi
    // jika kondisiKedua bernilai true
} else if(kondisiKetiga){
    // blok kode yang akan dieksekusi
    // jika kondisi ketiga bernilai true
} else {
    // blok kode yang akan dieksekusi
    // jika semua kondisi bernilai false
}
```

Contoh kode program percabangan if/else if/else :

```
fun main(args: Array<String>){

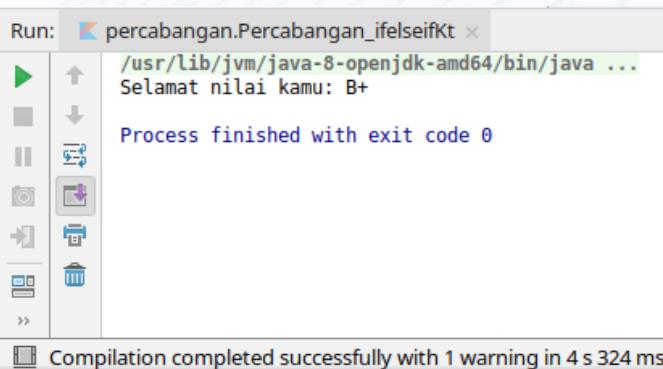
    val nilai: Int = 78
    var grade: String = ""

    if (nilai >= 90) grade = "A+"
    else if (nilai >= 80) grade = "A"
    else if (nilai >= 70) grade = "B+"
    else if (nilai >= 60) grade = "B"
    else if (nilai >= 50) grade = "C+"
    else if (nilai >= 40) grade = "C"
    else if (nilai >= 30) grade = "D"
    else if (nilai >= 20) grade = "E"
    else grade = "F"

    println("Selamat nilai kamu: $grade")

}
```

Output dari kode program di atas :



The screenshot shows a terminal window from an IDE. The title bar says "Run: percabangan.Percabangan\_ifelseifKt x". The main area displays the command "/usr/lib/jvm/java-8-openjdk-amd64/bin/java ...". Below it, the output "Selamat nilai kamu: B+" is shown in green. Further down, "Process finished with exit code 0" is displayed in blue. At the bottom of the window, a message indicates "Compilation completed successfully with 1 warning in 4 s 324 ms".

#### 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.

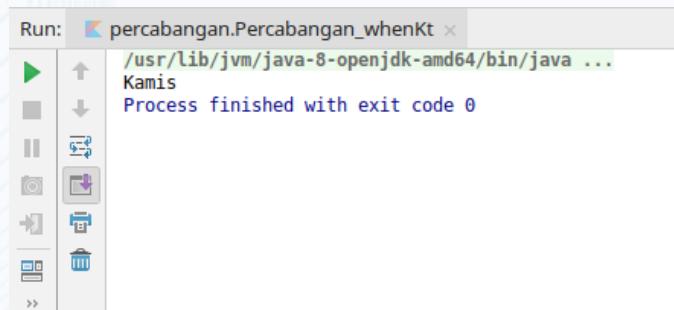
Sintaks dasar percabangan when :

```
when(variabel){  
    "nilai" -> {  
        // blok kode yang akan dieksekusi  
        // jika variabel sama dengan "nilai"  
    }  
    222 -> {  
        // blok kode yang akan dieksekusi  
        // jika variabel sama dengan 222  
    }  
    else {  
        // blok kode yang akan dieksekusi  
        // apabila variabel tidak sama dengan  
        // semua nilai di atas  
    }  
}
```

Contoh kode program percabangan when :

```
fun main(args: Array<String>){  
    var x: Int = 4  
  
    when (x) {  
        1 -> print("Senin")  
        2 -> print("Selasa")  
        3 -> print("Rabu")  
        4 -> print("Kamis")  
        5 -> print("Jum'at")  
        6 -> print("Sabtu")  
        7 -> print("Minggu")  
        else -> {  
            print("nomer hari salah!")  
        }  
    }  
}
```

Output dari kode program di atas :



```
Run: percabangan.Percabangan_whenKt
/usr/lib/jvm/java-8-openjdk-amd64/bin/java ...
Kamis
Process finished with exit code 0
```

## G. Perulangan

Struktur kontrol perulangan merupakan sebuah pernyataan dari Kotlin yang mengijinkan kita untuk mengeksekusi blok kode atau statement secara berulang-ulang sesuai dengan jumlah tertentu yang diinginkan. Berikut ini merupakan bentuk perulangan yang terdapat pada Kotlin :

### 1. Perulangan while loop

Pernyataan yang terdapat pada while loop akan dieksekusi berulang-ulang selama kondisi pada statement while bernilai true.

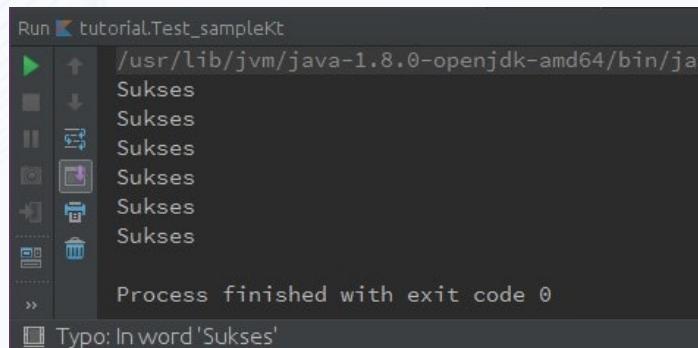
Sintaks dasar perulangan while loop :

```
while (kondisi) {
    // Pernyataan atau Statement
}
```

Contoh kode program perulangan while loop :

```
fun main(args: Array<String>){
    var angka = 0
    while(angka <= 5){
        println("Sukses")
        angka++
    }
}
```

Jika kita jalankan, pernyataan yang terdapat pada statement while akan terus dieksekusi hingga kondisi menjadi false, perintah angka++ digunakan agar nilai pada variabel angka terus bertambah 1 hingga kondisi pada while menjadi false dan program dihentikan.



```
Run < tutorial.Test_sampleKt
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java
Sukses
Sukses
Sukses
Sukses
Sukses
Sukses
Sukses
Process finished with exit code 0
Typo: In word 'Sukses'
```

Program tersebut akan mencetak kata "Sukses" sebanyak 5 kali. Perlu diingat jika bagian angka++ dihilangkan, akan menghasilkan pengulangan yang terus menerus (infinite loop). Pastikan agar kita memberikan pernyataan yang membuat pengulangan berhenti pada suatu kondisi.

## 2. Perulangan do-while loop

Statement do-while sama dengan statement while yaitu untuk mengeksekusi pernyataan berulang-ulang selama kondisi true.

Sintaks dasar perulangan do-while loop :

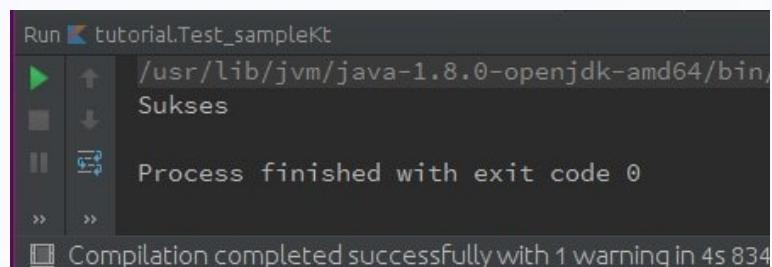
```
do {
    // Pernyataan atau Statement
} while (kondisi)
```

Perbedaannya, pernyataan pada statement do akan dieksekusi terlebih dahulu sebelum dievaluasi kondisinya pada statement while, jika kondisi pada statement while bernilai true, maka pernyataan pada do akan dieksekusi lagi secara berulang-ulang sampai bernilai false dan program dihentikan.

Contoh kode program perulangan do-while loop :

```
var angka = 0
do{
    println("Sukses")
    angka++
}while (angka < 5)
```

Jika kita jalankan, hasilnya akan sama dengan contoh program pada while loop.



```
Run < tutorial.Test_sampleKt
▶   /usr/lib/jvm/java-1.8.0-openjdk-amd64/bin,
  Sukses
||  Process finished with exit code 0
>> 
Compiling completed successfully with 1 warning in 4s 834
```

### 3. Perulangan for loop

Penggunaan for loop pada kotlin lebih simple dan mudah dibandingkan dengan Java, kita bisa menggunakan for untuk mengeksekusi pernyataan secara berulang-ulang, tetapi penggunaannya lebih simpel dibandingkan while dan do-while.

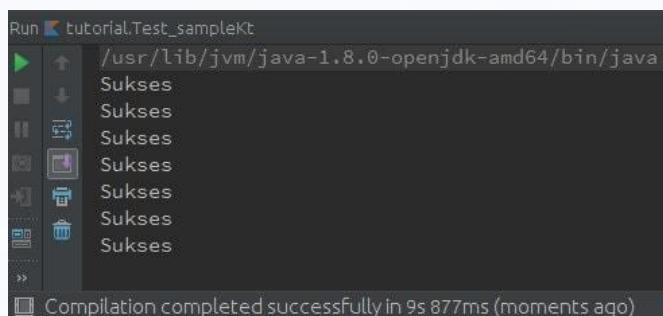
Sintaks dasar perulangan for loop :

```
for (variable(single/multiple variable) in expression) Pernyataan
atau
for (variable(single/multiple variable) in expression) {
    // Pernyataan
}
```

Pada contoh berikut ini, misalnya kita ingin mencetak kata "Sukses" sebanyak 7 kali, maka kita tidak memerlukan kondisi true, seperti pada while dan do-while, seperti ini :

```
fun main(args: Array<String>){  
    for (data in 1..7) println("Sukses")  
}
```

Output dari kode program di atas :



```
Run < tutorial.Test_sampleKt  
▶ /usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java  
Sukses  
Sukses  
Sukses  
Sukses  
Sukses  
Sukses  
Sukses  
Sukses  
Compilation completed successfully in 9s 877ms (moments ago)
```

Di sana terdapat sebuah variabel Integer bernama data, jadi program akan me-looping sesuai dengan range atau jumlah item pada expression yang diberikan, dan expression tersebut menjadi suatu nilai dari variabel item.

## H. Function

Jika sebelumnya pernah belajar seperti bahasa C#, PHP, atau Python, mungkin kita tidak asing dengan function atau method. Function sederhananya adalah sebuah kumpulan baris kode yang dapat kita panggil dengan sederhana (sering kali cukup dipanggil hanya dengan 1 baris saja) tanpa perlu menulis baris kode yang banyak berulang-ulang kali. Lalu bagaimana cara membuat function di Kotlin? Berikut ini merupakan contoh sederhananya:

```
fun namaFunction(nama: String): String {  
    return "Hello " + nama + "!"  
}  
  
fun main(args: Array<String>) {  
    println(namaFunction("World"))  
}
```

Bagaimana hasilnya ketika dijalankan?

```
Compilation completed successfully
```

```
Hello World!
```

Untuk membuat sebuah function, kita perlu mendefinisikan ‘fun’. Lalu diikuti dengan nama function. Penulisan nama function harus digabung, gunakan camel case seperti ‘namaFunctionKamuBebas’ untuk memudahkan orang lain dalam membaca kode kita nantinya.

Setelah itu diikuti dengan parameter. Parameter adalah sebuah data yang dibutuhkan oleh function untuk nantinya diolah di dalam function. Untuk menuliskan parameter, tulis nama variabel dari parameter terlebih dahulu. Penamaan variabel digunakan untuk memanggil variabel di dalam function. Setelah itu diikuti dengan tipe datanya, bisa String ataupun menggunakan tipe data yang lain, disesuaikan dengan kebutuhan.

Setelah itu, diikuti dengan return dari function. Return dari function ini berguna untuk mendefinisikan apakah function kita memiliki return atau tidak. Jika memiliki return, maka kita perlu mendefinisikan tipe data apa yang dihasilkan oleh function yang kita buat. Contohnya di function

namaParameter, pada kode di atas kita menggunakan String sebagai return dari function, karena kita ingin mengembalikan nilai String yang telah diolah. Setelah menuliskan return dari function, kita buka bracket untuk menentukan isi dari function kita apa. Jika function kita memiliki return, maka pastikan untuk memanggil sintaks return di akhir dari functionnya. Untuk memanggil function sendiri, kita bisa memanggil namanya diikuti dengan buka kurung berisi data yang ingin kita gunakan di dalam function (arguments).

## 1. Parameter

Sebelumnya kita telah menyinggung sedikit apa itu parameter. Parameter adalah sebuah data yang dibutuhkan oleh function untuk nantinya diolah di dalam function. Ketika membuat function, kita tidak hanya dapat menggunakan 1 parameter saja. Kita juga bisa menggunakan lebih dari 2 parameter. Kita juga bisa tidak menggunakan parameter sama sekali. Gunakanlah parameter sesuai dengan kebutuhan masing-masing. Berikut contohnya :

Function yang menggunakan 2 parameter, yakni function operasi aritmatika dasar.

```
fun tambah(angka1: Int, angka2: Int): Int {}
```

Function yang menggunakan 3 parameter, yakni function untuk menentukan angka tertinggi.

```
fun angkaTertinggi(angka1: Int, angka2: Int, angka3: Int): Int {}
```

Function yang tidak menggunakan parameter, yakni function untuk melakukan pengacakan nomor.

```
fun acakNomor(): Int {}
```

dan masih banyak lagi contohnya. Selain itu, kita dapat menggunakan berbagai tipe data yang telah dijelaskan sebelumnya dari String, Int, Double, Boolean, hingga Array. Berikut contohnya :

```
fun menggunakanBanyakTipeData(angka: Int, huruf: Char, kata: String,  
Desimal: Double, array: Array<Double>): Int {}
```

## 2. Parameter dengan Default Value

Di Kotlin, kita juga dapat mengisi default value dari parameter. Maksudnya bagaimana? Kita dapat memberikan opsi pada function kita. Jika saat memanggil function, lalu kita tidak mengirimkan data sesuai jumlah data yang dibutuhkan di parameter, maka akan terjadi error. Untuk menghindari error tersebut, kita dapat menggunakan default value sehingga meskipun data tidak dikirimkan sesuai jumlah parameter, function tetap akan berjalan dengan baik. Contohnya dalam function tambah, kita membutuhkan 2 data untuk ditambahkan. Namun bagaimana jika kita tidak mengirimkan 2 data tersebut? Akan terjadi error, karena tidak ada data ditambah tidak ada data maka tidak dapat melakukan penjumlahan. Berikut cara menggunakan default value untuk menghindari error:

```
fun tambahDenganDefaultValue(angka1: Int = 1, angka2: Int = 1): Int {}
```

Setelah mendefinisikan tipe data dari parameter, kita dapat mengisi default value dari parameter tersebut. Berikut contoh programnya:

```
fun tambahDenganDefaultValue(angka1: Int = 1, angka2: Int = 1): Int {  
    return angka1 + angka2  
}  
  
fun main(args: Array<String>) {  
    println(tambahDenganDefaultValue())  
}
```

Berikut hasilnya:

```
Compilation completed successfully  
2
```

Jadi meskipun kita tidak mengirimkan data sesuai dengan jumlah parameter, kita tetap dapat menggunakan function tersebut tanpa ada masalah.

### 3. Argument

Argument juga sempat disinggung sebelumnya, jika Parameter digunakan untuk mendefinisikan bahwa sebuah function membutuhkan sebuah data tertentu untuk dikirimkan ke dalam function. Maka, argument adalah sebuah data yang dikirimkan ke function.

```
function [parameter 1][parameter 2]
```

```
memanggilFunction(argumen 1,[argumen 2])
```

Gambaran perbedaan antara parameter dengan argument

Di Kotlin, misalkan kita memiliki sebuah function yang memiliki 5 parameter yang memiliki default value. Lalu kita ingin memanggil function tersebut, dan menggunakan argument kedua dan kelima saja. Pada bahasa lain, kita harus urut memanggil argumennya satu-per-satu, seperti berikut:

```
fun contohBahasaLain(param1: String = "a", param2: String = "b", param3:  
String = "c", param4: String = "d", param5: String = "e")
```

Lalu misalkan kita ingin memanggil function tersebut, namun kita ingin mengirimkan data sehingga parameter ke-3 tidak bernilai “c” lagi tetapi “x”. Maka ketika memanggil function menjadi seperti ini dalam bahasa lain:

```
val memanggilFunction = contohBahasaLain(arg1, arg2, arg3)
```

Dalam bahasa lain, ketika kita hanya ingin mengirimkan data ke-3, kita harus mengirimkan juga data dari parameter sebelumnya. Sedangkan di dalam Kotlin terdapat named arguments. Artinya, ketika nama variabel parameter kita sama dengan nama variabel argument kita, meskipun kita hanya mengubah data parameter ke-3, kita tidak perlu menuliskan data parameter ke-1 dan ke-2. Contohnya berikut masih tetap menggunakan function contohBahasaLain:

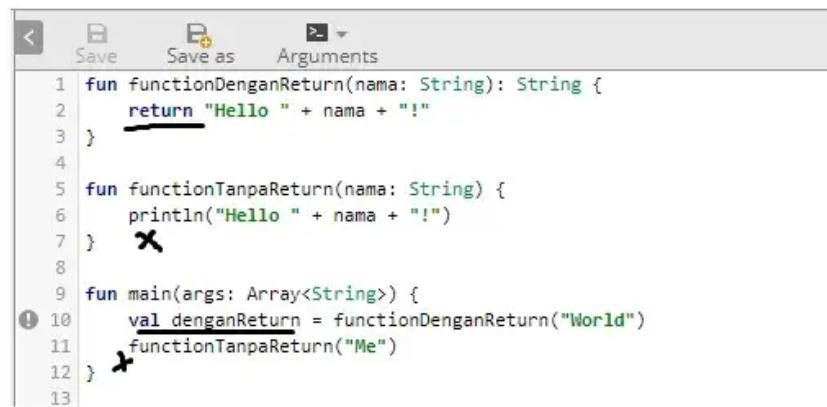
```
val memanggilNamedArgument = contohBahasaLain(param3 = "x")
```

Kita cukup menggunakan 1 argument saja, namun nama argument tersebut harus sama dengan nama variabel dari parameter.

#### 4. Return dari Function

Pada saat membahas function, kita telah berkenalan dengan return dari function. Sederhananya, return dari function membantu Kotlin mendefinisikan apakah Function kita memiliki return value atau tidak. Jika function kita memiliki return value, maka kita perlu mendefinisikan tipe data dari return value tersebut.

Perbedaan sederhana antara function yang memiliki return value dan tidak dapat dilihat pada contoh kode berikut:



```
Save Save as Arguments
1 fun functionDenganReturn(nama: String): String {
2     return "Hello " + nama + "!"
3 }
4
5 fun functionTanpaReturn(nama: String) {
6     println("Hello " + nama + "!")
7 }
8
9 fun main(args: Array<String>) {
10    val denganReturn = functionDenganReturn("World")
11    functionTanpaReturn("Me")
12 }
13
```

The screenshot shows a code editor window with a toolbar at the top labeled 'Save', 'Save as', and 'Arguments'. The code area contains three functions: 'functionDenganReturn' which returns a string, 'functionTanpaReturn' which prints a string to the console, and the 'main' function which calls both. The 'functionDenganReturn' function has a red underline under the 'return' statement, while the 'functionTanpaReturn' function has a red 'X' over its entire body. The 'main' function's call to 'functionTanpaReturn' also has a red 'X' over it.

Perbedaan antara function yang memiliki return value dan tanpa return value adalah, yang pertama, untuk function yang memiliki return value maka di akhir function kita perlu mendefinisikan return dengan nilai apa yang ingin kita kirimkan kembali ke baris yang memanggil function tersebut, yakni baris ke 10. Sedangkan, pada function yang tidak memiliki return value, kita tidak perlu menuliskan return di dalam function tersebut.

Perbedaan kedua, pada function yang menggunakan return value, kita harus terlebih dahulu menampung atau menyimpan hasil dari pemanggilan function tersebut. Karena function yang memiliki return value akan mengembalikan sebuah nilai atau value hasil dari pengolahan di dalam function. Sementara, pada function yang tidak memiliki return value, kita tidak perlu membuat variabel untuk menyimpan nilai yang dihasilkan dari function tersebut, karena pada dasarnya function yang tidak memiliki return value otomatis tidak mengembalikan nilai apa-apa. Function yang tidak memiliki return value, sering kali disebut sebagai void. Sementara untuk function yang memiliki return value, kita dapat menggunakan berbagai macam tipe data sebagai return value dari function tersebut. Kita dapat menggunakan baik yang sering kita gunakan seperti String, Int, atau bahkan kita dapat menggunakan Array dan sebagainya.

## References

<https://developer.android.com/topic/libraries/view-binding?hl=id>

<https://developer.android.com/studio/build?hl=id>

<https://www.niagahoster.co.id/blog/kotlin-adalah/>

<https://www.petanikode.com/kotlin-variabel-tipe-data/>

<https://www.petanikode.com/kotlin-percabangan/>

<https://www.wildantechnoart.net/2017/06/looping-perulangan-kotlin-do-while-for-loops.html>

<https://medium.com/@ridhofebriansa/belajar-dasar-kotlin-function-a47b6c57e3fe>