WHAT ?

```
fun main() {
    println("Hello, world!")
    // Hello, world!
}
```

dalam Kotlin:
- fun digunakan untuk mendeklarasikan sebuah fungsi
- fungsi main() adalah tempat program Anda dimulai
- badan dari sebuah fungsi ditulis di dalam kurung kurawal {}
- fungsi println() dan print() mencetak argumen mereka ke output standar

Variables

Semua program harus dapat menyimpan data, dan variabel membantu Anda untuk melakukan hal tersebut. Di Kotlin, Anda dapat mendeklarasikan:
- variabel read-only dengan ***val***
- variabel yang dapat diubah dengan ***var***

Untuk menetapkan nilai, gunakan operator penugasan **=**

String templates

Akan sangat berguna untuk mengetahui cara mencetak isi variabel ke output standar. Anda dapat melakukan ini dengan template string. Anda dapat menggunakan ekspresi template untuk mengakses data yang disimpan dalam variabel dan objek lain, dan mengubahnya menjadi string. Nilai string adalah urutan karakter dalam tanda kutip ganda ". Ekspresi template selalu dimulai dengan tanda dolar **$**.
Untuk mengevaluasi sepotong kode dalam ekspresi template, letakkan kode di dalam kurung kurawal {} setelah tanda dolar $.

```
val customers = 10
println("There are $customers customers")
```

TIPE DATA DASAR

Category	Basic types
Integers	`Byte`, `Short`, `Int`, `Long`
Unsigned integers	`UByte`, `UShort`, `UInt`, `ULong`
Floating-point numbers	`Float`, `Double`
Booleans	`Boolean`
Characters	`Char`
Strings	`String`

```
// Variable dideklarasikan tanpa inisialisasi
val d: Int
// Variable dengan inisialisasi
d = 3
// Variable jenis explicit dan di inisialisasi
val e: String = "hello"
```

COLLECTION

Collection type	Description
Lists	Ordered collections of items
Sets	Unique unordered collections of items
Maps	Sets of key-value pairs where keys are unique and map to only one value

List

List menyimpan item sesuai urutan penambahannya, dan memungkinkan adanya item duplikat.
Untuk membuat List Read-only (List), gunakan fungsi listOf().
Untuk membuat List yang dapat diubah (MutableList), gunakan fungsi mutableListOf().
Ketika membuat List, Kotlin dapat menyimpulkan jenis item yang disimpan. Untuk mendeklarasikan tipe secara eksplisit, tambahkan tipe di dalam tanda kurung siku <> setelah deklarasi List:

```
// Read only list
val readOnlyShapes = listOf("triangle", "square", "circle")
println(readOnlyShapes)
// [triangle, square, circle]

// Mutable list with explicit type declaration
val shapes: MutableList<String> = mutableListOf("triangle", "square", "circle")
println(shapes)
// [triangle, square, circle]
```

Untuk mencegah modifikasi yang tidak diinginkan, Anda dapat memperoleh View yang bersifat Read-Only dari mutable List dengan menetapkannya sebagai List:

```
val shapes: MutableList<String> = mutableListOf("triangle", "square", "circle")
val shapesLocked: List<String> = shapes
```

List diurutkan sehingga untuk mengakses item dalam List, gunakan operator akses terindeks []:
Untuk mendapatkan item pertama atau terakhir dalam List, gunakan fungsi .first() dan .last():
Untuk mendapatkan jumlah item dalam List, gunakan fungsi .count():
Untuk memeriksa apakah sebuah item ada di dalam List, gunakan operator in:
Untuk menambah atau menghapus item dari mutable-List, gunakan fungsi .add() dan .remove():

Set

Sementara List diurutkan dan memungkinkan item duplikat, set tidak diurutkan dan hanya menyimpan item unik.
Untuk membuat set yang Read-only (Set), gunakan fungsi setOf().
Untuk membuat set yang dapat diubah (MutableSet), gunakan fungsi mutableSetOf().
Ketika membuat set, Kotlin dapat menyimpulkan jenis item yang disimpan. Untuk mendeklarasikan tipe secara eksplisit, tambahkan tipe di dalam tanda kurung siku <> setelah deklarasi Set:

```
// Read-only set
val readOnlyFruit = setOf("apple", "banana", "cherry", "cherry")
// Mutable set with explicit type declaration
val fruit: MutableSet<String> = mutableSetOf("apple", "banana", "cherry", "cherry")

println(readOnlyFruit)
// [apple, banana, cherry]
```

Untuk mencegah modifikasi yang tidak diinginkan, Anda dapat memperoleh View yang bersifat Read-Only dari mutable Set dengan menetapkannya sebagai Set:

```
val fruit: MutableSet<String> = mutableSetOf("apple", "banana", "cherry", "cherry")
val fruitLocked: Set<String> = fruit
```

Untuk mendapatkan jumlah item dalam satu set, gunakan fungsi .count():
Untuk memeriksa apakah sebuah item ada di dalam sebuah set, gunakan operator in:
Untuk menambah atau menghapus item dari set yang dapat diubah, gunakan fungsi .add() dan .remove():

Map

Maps menyimpan item data sebagai pasangan key-value.

Untuk membuat map yang read-only (Map), gunakan fungsi mapOf().

Untuk membuat map yang dapat diubah (MutableMap), gunakan fungsi mutableMapOf().

Ketika membuat map, Kotlin dapat menyimpulkan jenis item yang disimpan. Untuk mendeklarasikan tipe secara eksplisit, tambahkan tipe kunci dan nilai dalam tanda kurung siku <> setelah deklarasi map. Sebagai contoh: MutableMap<String, Int>. Kunci memiliki tipe String dan nilai memiliki tipe Int.
Cara termudah untuk membuat map adalah dengan menggunakan 'to' di antara setiap kunci dan nilai terkait:

```
// Read-only map
val readOnlyJuiceMenu = mapOf("apple" to 100, "kiwi" to 190, "orange" to 100)
println(readOnlyJuiceMenu)
// {apple=100, kiwi=190, orange=100}

// Mutable map with explicit type declaration
val juiceMenu: MutableMap<String, Int> = mutableMapOf("apple" to 100, "kiwi" to 190,
"orange" to 100)
println(juiceMenu)
// {apple=100, kiwi=190, orange=100}
```

Untuk mencegah modifikasi yang tidak diinginkan, dapatkan View raed-only dari mutable map dengan menetapkannya sebagai map:

```
val juiceMenu: MutableMap<String, Int> = mutableMapOf("apple" to 100, "kiwi" to 190,
"orange" to 100)
val juiceMenuLocked: Map<String, Int> = juiceMenu
```

Untuk mengakses nilai pada map, gunakan operator akses yang diindeks [] dengan 'key'nya:

```
// Read-only map
val readOnlyJuiceMenu = mapOf("apple" to 100, "kiwi" to 190, "orange" to 100)
println("The value of apple juice is: ${readOnlyJuiceMenu["apple"]}")
// The value of apple juice is: 100
```

Untuk mendapatkan jumlah item dalam Map, gunakan fungsi .count():

Untuk menambah atau menghapus item dari Map yang dapat diubah, gunakan fungsi .put() dan .remove():
Untuk memeriksa apakah kunci tertentu sudah disertakan dalam Map, gunakan fungsi .containsKey():
Untuk mendapatkan koleksi kunci atau nilai dari sebuah Map, gunakan properti kunci dan nilai:
Untuk memeriksa apakah sebuah kunci atau nilai ada di dalam Map, gunakan operator in:

Conditional expressions

Kotlin menyediakan if dan when untuk memeriksa ekspresi bersyarat.

If

Untuk menggunakan if, tambahkan ekspresi kondisional di dalam tanda kurung () dan tindakan yang akan dilakukan jika hasilnya benar di dalam tanda kurung kurawal {}:

```
val d: Int
val check = true

if (check) {
    d = 1
} else {
    d = 2
}

println(d)
```

When

Gunakan when ketika Anda memiliki ekspresi bersyarat dengan beberapa cabang. when dapat digunakan baik sebagai pernyataan maupun ekspresi.
Berikut adalah contoh penggunaan when sebagai pernyataan:
- Tempatkan ekspresi kondisional di dalam tanda kurung () dan tindakan yang harus dilakukan di dalam tanda kurung kurawal {}.
- Gunakan -> di setiap cabang untuk memisahkan setiap kondisi dari setiap tindakan.

Contoh 1:

```
when (obj) {
    // Checks whether obj equals to "1"
    "1" -> println("One")
    // Checks whether obj equals to "Hello"
    "Hello" -> println("Greeting")
    // Default statement
```

```
    else -> println("Unknown")
}
// Greeting
```

Contoh 2 :

```
val obj = "Hello"

val result = when (obj) {
    // If obj equals "1", sets result to "one"
    "1" -> "One"
    // If obj equals "Hello", sets result to "Greeting"
    "Hello" -> "Greeting"
    // Sets result to "Unknown" if no previous condition is satisfied
    else -> "Unknown"
}
println(result)
// Greeting
```

Ranges

Sebelum membahas tentang perulangan, ada baiknya kita mengetahui cara membuat rentang untuk perulangan.

Cara paling umum untuk membuat rentang di Kotlin adalah dengan menggunakan operator .... Sebagai contoh, 1..4 setara dengan 1, 2, 3, 4.

Untuk mendeklarasikan sebuah range yang tidak menyertakan nilai akhir, gunakan operator ..<. Misalnya, 1..<4 setara dengan 1, 2, 3.

Untuk mendeklarasikan rentang dalam urutan terbalik, gunakan downTo. Misalnya, 4 downTo 1 setara dengan 4, 3, 2, 1.

Untuk mendeklarasikan rentang yang bertambah dalam langkah yang bukan 1, gunakan langkah dan nilai kenaikan yang Anda inginkan. Misalnya, 1..5 langkah 2 setara dengan 1, 3, 5.

Anda juga dapat melakukan hal yang sama dengan rentang Char:

- 'a'...d' setara dengan 'a', 'b', 'c', 'd'

- 'z' downTo 's' langkah 2 setara dengan 'z', 'x', 'v', 't'

Loops

Dua struktur perulangan yang paling umum dalam pemrograman adalah for dan while. Gunakan for untuk mengulang serangkaian nilai dan melakukan suatu tindakan. Gunakan while untuk melanjutkan tindakan hingga kondisi tertentu terpenuhi.

For

Anda dapat membuat perulangan for yang mengulang angka 1 hingga 5 dan mencetak angka tersebut setiap kali.

Tempatkan iterator dan rentang di dalam tanda kurung () dengan kata kunci in. Tambahkan tindakan yang ingin Anda selesaikan di dalam tanda kurung kurawal {}:

```
for (number in 1..5) {
   // number is the iterator and 1..5 is the range
   print(number)
}
// 12345
```

While

while dapat digunakan dalam dua cara:
- Untuk mengeksekusi blok kode selama ekspresi kondisional bernilai benar (while)
- Untuk mengeksekusi blok kode terlebih dahulu dan kemudian memeriksa ekspresi kondisional. (do-while)

Dalam kasus penggunaan pertama (while):
- Nyatakan ekspresi kondisional untuk perulangan perulangan Anda untuk dilanjutkan di dalam tanda kurung ().
- Tambahkan tindakan yang ingin Anda selesaikan di dalam tanda kurung kurawal {}.

Functions

Anda bisa mendeklarasikan fungsi Anda sendiri di Kotlin dengan menggunakan kata kunci fun.

Dalam Kotlin:
- parameter fungsi ditulis di dalam tanda kurung ().
- setiap parameter harus memiliki tipe, dan beberapa parameter harus dipisahkan dengan koma ,.

- tipe pengembalian dituliskan setelah tanda kurung () fungsi, dipisahkan dengan tanda titik dua :.
- tubuh fungsi ditulis dalam kurung kurawal {}.
- kata kunci return digunakan untuk keluar atau mengembalikan sesuatu dari sebuah fungsi.

Jika sebuah fungsi tidak mengembalikan sesuatu yang berguna, tipe return dan kata kunci return dapat dihilangkan. Pelajari lebih lanjut tentang hal ini di Fungsi tanpa pengembalian.

Dalam contoh berikut:
- x dan y adalah parameter fungsi.
- x dan y memiliki tipe Int.
- tipe hasil fungsi adalah Int.
- fungsi mengembalikan jumlah x dan y saat dipanggil.

```
fun sum(x: Int, y: Int): Int {
    return x + y
}

fun main() {
    println(sum(1, 2))
    // 3
```

Named arguments

Untuk kode yang ringkas, ketika memanggil fungsi, Anda tidak perlu menyertakan nama parameter. Namun, menyertakan nama parameter akan membuat kode Anda lebih mudah dibaca. Ini disebut menggunakan argumen bernama. Jika Anda menyertakan nama parameter, maka Anda dapat menulis parameter dalam urutan apa pun.

Pada contoh berikut, template string ($) digunakan untuk mengakses nilai parameter, mengonversinya menjadi tipe String, dan kemudian menggabungkannya menjadi sebuah string untuk dicetak.

```
fun printMessageWithPrefix(message: String, prefix: String = "Info") {
    println("[$prefix] $message")
}

fun main() {
    // Uses named arguments with swapped parameter order
    printMessageWithPrefix(prefix = "Log", message = "Hello")
```

```
    // [Log] Hello
}
```

Default parameter values

Anda dapat menentukan nilai default untuk parameter fungsi Anda. Parameter apa pun dengan nilai default dapat dihilangkan saat memanggil fungsi Anda. Untuk mendeklarasikan nilai default, gunakan operator penugasan = setelah tipe:

```
fun printMessageWithPrefix(message: String, prefix: String = "Info") {
    println("[$prefix] $message")
}

fun main() {
    // Function called with both parameters
    printMessageWithPrefix("Hello", "Log")
    // [Log] Hello

    // Function called only with message parameter
    printMessageWithPrefix("Hello")
    // [Info] Hello

    printMessageWithPrefix(prefix = "Log", message = "Hello")
    // [Log] Hello
}
```

Functions without return

Jika fungsi Anda tidak mengembalikan nilai yang berguna, maka tipe kembaliannya adalah Unit. Unit adalah tipe dengan hanya satu nilai - Unit. Anda tidak perlu mendeklarasikan bahwa Unit dikembalikan secara eksplisit dalam badan fungsi Anda. Ini berarti Anda tidak perlu menggunakan kata kunci return atau mendeklarasikan tipe pengembalian:

```
fun printMessage(message: String) {
    println(message)
    // `return Unit` or `return` is optional
}
```

```
fun main() {
    printMessage("Hello")
    // Hello
}
```

Lambda expressions

Kotlin memungkinkan Anda untuk menulis kode yang lebih ringkas untuk fungsi-fungsi dengan menggunakan ekspresi lambda.
Sebagai contoh, fungsi uppercaseString() berikut ini:

```
fun uppercaseString(string: String): String {
    return string.uppercase()
}
fun main() {
    println(uppercaseString("hello"))
    // HELLO
}
```

Dapat ditulis dalam ekspresi lambda sbb :

```
fun main() {
    println({ string: String -> string.uppercase() }("hello"))
    // HELLO
}
```

Class

Kotlin mendukung pemrograman berorientasi objek dengan kelas dan objek. Objek berguna untuk menyimpan data dalam program Anda. Kelas memungkinkan Anda untuk mendeklarasikan sekumpulan karakteristik untuk sebuah objek. Ketika Anda membuat objek dari sebuah kelas, Anda dapat menghemat waktu dan tenaga karena Anda tidak perlu mendeklarasikan karakteristik ini setiap saat.

Untuk mendeklarasikan sebuah kelas, gunakan kata kunci class:

```
class Customer
```

Properties

Karakteristik objek kelas dapat dideklarasikan dalam properti. Anda dapat mendeklarasikan properti untuk sebuah kelas:

- Di dalam tanda kurung () setelah nama kelas.

```
class Contact(val id: Int, var email: String)
```

Create instance

Untuk membuat objek dari sebuah kelas, Anda mendeklarasikan sebuah instance kelas menggunakan konstruktor.

Secara default, Kotlin secara otomatis membuat konstruktor dengan parameter yang dideklarasikan di header kelas.

Sebagai contoh:

```
class Contact(val id: Int, var email: String)
fun main() {
    val contact = Contact(1, "mary@gmail.com")
}
```

Access properties

Untuk mengakses properti dari sebuah instance, tulis nama properti setelah nama instance yang ditambahkan dengan titik . :

```
fun main() {
    val contact = Contact(1, "mary@gmail.com")

    // Prints the value of the property: email
    println(contact.email)
    // mary@gmail.com

    // Updates the value of the property: email
    contact.email = "jane@gmail.com"

    // Prints the new value of the property: email
    println(contact.email)
    // jane@gmail.com
}
```

Member functions

Selain mendeklarasikan properti sebagai bagian dari karakteristik objek, Anda juga dapat mendefinisikan perilaku objek dengan fungsi anggota.

Di Kotlin, fungsi anggota harus dideklarasikan di dalam badan kelas. Untuk memanggil fungsi anggota pada sebuah instance, tuliskan nama fungsi setelah nama instance yang ditambahkan dengan titik.. Sebagai contoh:

```
class Contact(val id: Int, var email: String) {
    fun printId() {
        println(id)
    }
}

fun main() {
    val contact = Contact(1, "mary@gmail.com")
    // Calls member function printId()
    contact.printId()
    // 1
}
```

Data classes

Kotlin memiliki kelas data yang sangat berguna untuk menyimpan data. Kelas data memiliki fungsi yang sama dengan kelas, namun secara otomatis dilengkapi dengan fungsi anggota tambahan. Fungsi-fungsi anggota ini memungkinkan Anda untuk dengan mudah mencetak instance ke output yang dapat dibaca, membandingkan instance dari sebuah kelas, menyalin instance, dan banyak lagi. Karena fungsi-fungsi ini tersedia secara otomatis, Anda tidak perlu menghabiskan waktu untuk menulis kode boilerplate yang sama untuk setiap kelas Anda.

Untuk mendeklarasikan kelas data, gunakan kata kunci data:

```
data class User(val name: String, val id: Int)
```

Fungsi anggota yang telah ditentukan sebelumnya dari kelas data yang paling berguna adalah:

Function	Description
`.toString()`	Mencetak string yang dapat dibaca dari instance kelas dan propertinya.
`.equals()` or ==	Membandingkan instance dari suatu kela.
`.copy()`	Membuat contoh kelas dengan menyalin kelas lain, yang mungkin memiliki beberapa properti berbeda.

Print as string

Untuk mencetak string yang dapat dibaca dari instance kelas, Anda dapat secara eksplisit memanggil fungsi .toString(), atau menggunakan fungsi cetak (println() dan print()) yang secara otomatis memanggil .toString() untuk Anda:

```
// Secara otomatis menggunakan fungsi toString() agar output mudah dibaca

println(user)

// User (nama = Alex, id = 1)
```

Ini sangat berguna ketika melakukan debug atau membuat log.

Compare instances

Untuk membandingkan instance kelas data, gunakan operator persamaan ==:

```
val user = User("Alex", 1)
val secondUser = User("Alex", 1)
val thirdUser = User("Max", 2)

// Membandingkan User dengan User kedua
println("user == secondUser: ${user == secondUser}")
// user == secondUser: true

// Membandingkan User dengan User ketiga
println("user == thirdUser: ${user == thirdUser}")
// user == thirdUser: false
```

Copy instance

Untuk membuat salinan persis dari instance kelas data, panggil fungsi .copy() pada instance.

Untuk membuat salinan instance kelas data dan mengubah beberapa properti, panggil fungsi .copy() pada instance dan tambahkan nilai pengganti untuk properti sebagai parameter fungsi.

Sebagai contoh:

```
val user = User("Alex", 1)
val secondUser = User("Alex", 1)
val thirdUser = User("Max", 2)

// Membuat salinan yang tepat dari User
println(user.copy())
// User (nama = Alex, id = 1)

// Membuat salinan User dengan nama: "Max"
println(user.copy("Max"))
// User(nama=Max, id=1)

// Membuat salinan User dengan id: 3
println(user.copy(id = 3))
// User (nama = Alex, id = 3)
```

Null safety

Di Kotlin, nilai null bisa saja menjadi nilai yang tidak berarti. Untuk membantu mencegah masalah dengan nilai null dalam program Anda, Kotlin memiliki pengaman null. Null safety mendeteksi potensi masalah dengan nilai null pada saat kompilasi, bukan pada saat dijalankan.

Null safety adalah kombinasi dari fitur-fitur yang memungkinkan Anda untuk:
- secara eksplisit menyatakan kapan nilai null diperbolehkan dalam program Anda.
- memeriksa nilai null.
- menggunakan pemanggilan yang aman ke properti atau fungsi yang mungkin berisi nilai null.
- mendeklarasikan tindakan yang harus diambil jika nilai null terdeteksi.

Nullable types

Kotlin mendukung tipe nullable yang memungkinkan tipe yang dideklarasikan memiliki nilai null. Secara default, sebuah tipe tidak diperbolehkan menerima nilai null. Tipe nullable dideklarasikan dengan menambahkan secara eksplisit ? setelah deklarasi tipe.

Sebagai contoh:

```
fun main() {
    // neverNull has String type
    var neverNull: String = "This can't be null"

    // Throws a compiler error
    neverNull = null

    // nullable has nullable String type
    var nullable: String? = "You can keep a null here"

    // This is OK
    nullable = null

    // By default, null values aren't accepted
    var inferredNonNull = "The compiler assumes non-nullable"

    // Throws a compiler error
    inferredNonNull = null

    // notNull doesn't accept null values
    fun strLength(notNull: String): Int {
        return notNull.length
    }

    println(strLength(neverNull)) // 18
    println(strLength(nullable))  // Throws a compiler error
}
```

Check for null values

Anda dapat memeriksa keberadaan nilai null di dalam ekspresi bersyarat. Pada contoh berikut, fungsi describeString() memiliki pernyataan if yang memeriksa apakah maybeString tidak bernilai null dan apakah panjangnya lebih besar dari nol:

```
fun describeString(maybeString: String?): String {
    if (maybeString != null && maybeString.length > 0) {
        return "String of length ${maybeString.length}"
    } else {
        return "Empty or null string"
    }
}

fun main() {
    var nullString: String? = null
    println(describeString(nullString))
    // Empty or null string
}
```

Use safe calls

Untuk mengakses properti objek dengan aman yang mungkin berisi nilai null, gunakan operator safe call ?... Operator safe call mengembalikan nilai null jika properti objek bernilai null. Hal ini berguna jika Anda ingin menghindari adanya nilai null yang memicu kesalahan dalam kode Anda.

Pada contoh berikut, fungsi lengthString() menggunakan pemanggilan aman untuk mengembalikan panjang string atau null:

```
fun lengthString(maybeString: String?): Int? = maybeString?.length

fun main() {
    var nullString: String? = null
    println(lengthString(nullString))
    // null
}
```

Use Elvis operator

Anda dapat memberikan nilai default untuk dikembalikan jika nilai nol terdeteksi dengan menggunakan operator Elvis ?:.

Tuliskan di sisi kiri operator Elvis apa yang harus diperiksa untuk nilai nol. Tulis di sisi kanan operator Elvis apa yang harus dikembalikan jika nilai null terdeteksi.

Pada contoh berikut, nullString adalah null sehingga pemanggilan yang aman untuk mengakses properti panjang mengembalikan nilai null. Hasilnya, operator Elvis mengembalikan nilai 0:

```
fun main() {
    var nullString: String? = null
    println(nullString?.length ?: 0)
    // 0
}
```