Akhmad Fauzi (PC-11)

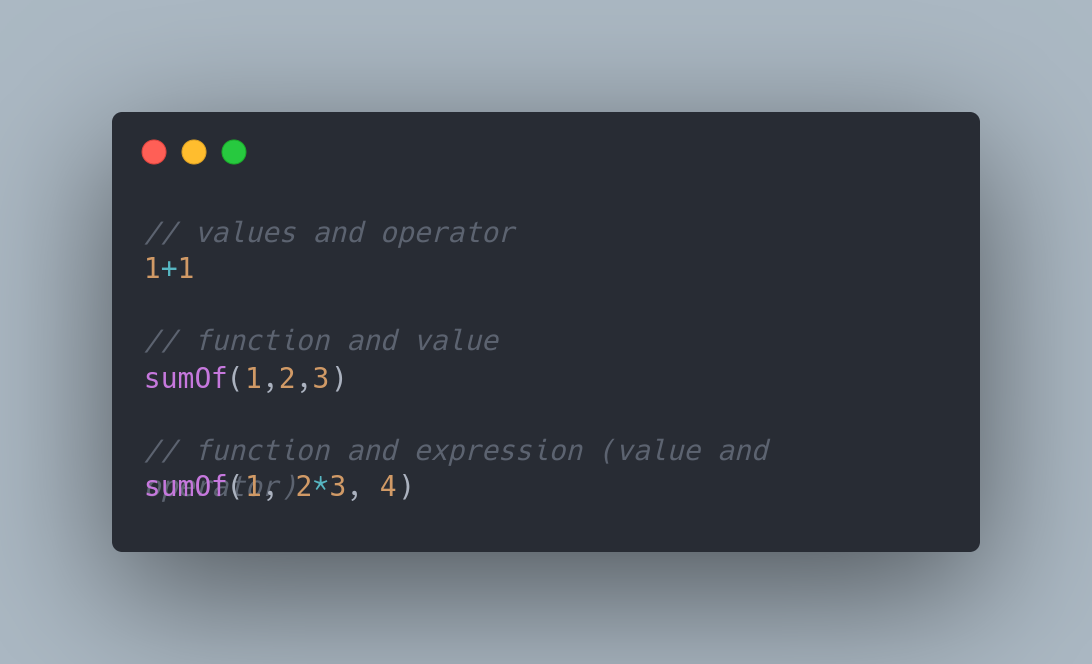
*Mobile Application and Technology*

**LAPORAN MATERI**

**MENULIS KODE DENGAN PRINSIP SESUAI GUIDELINES DAN BEST PRACTICES**

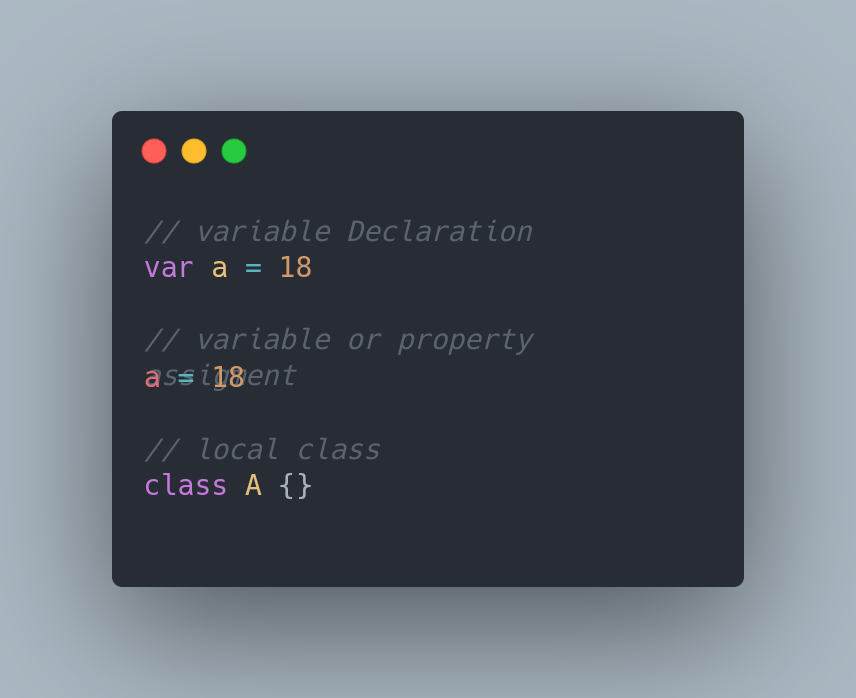
“Menulis Kode dengan Prinsip sesuai *Guidelines* dan *Best Practice*” yang dibawakan oleh instruktur (Pak Agung), dimana berisi pengetahuan dan pedoman dasar dalam menggunakan bahasa pemograman Kotlin.

*Guidelines* suatu bahasa pemograman merupakan pedoman atau panduan penggunaan bahasa pemograman untuk mengembangkan suatu *product* dimana berisi *basic syntax, keyword*, dan konsep penggunaan bahasa pemograman tersebut. Selain *guidelines, best practices* juga dapat dimanfaatkan sebagai pedoman pemograman yang ditetapkan oleh suatu organisasi, instansi, atau kelompok tertentu agar *source codes-*nya lebih mudah dipahami dan dapat mengefisiensikan pemograman (seperti dapat mengurangi tingkat kompleksitas program, menghemat penyimpanan, dan mempercepat proses komputasi program). Pada kesempatan kali ini membahas bahasa pemograman Kotlin untuk membuat sebuah *mobile application.* *Guidelines* bahasa pemograman dapat ditemukan di dokumentasi pada situs resmi bahasa pemograman terkait, untuk Kotlin dapat ditemukan pada kotlinlang.org/docs. Sedangkan *best practices* yang terapkan sesuai dengan pedoman dari BBPLK Bekasi.



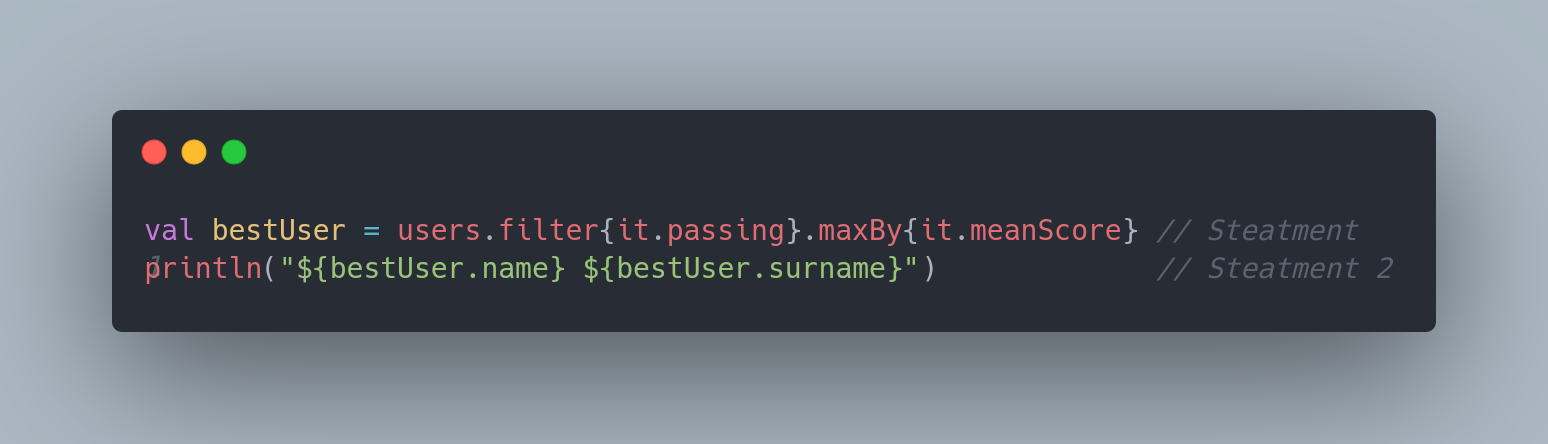
Gambar 1. Contoh expressions dalam Kotlin

Suatu *source codes* terdiri dari beberapa *expressions* dan *statements* yang setiap bahasa pemograman memiliki definisi yang berbeda-beda.Dalam Kotlin, *expression* merupakan kombinasi antara *values, variables, operators,* dan *functions* yang menghasilkan suatu *value* atau *unit* seperti pada gambar 1. Namun tidak semuanya dapat disebut sebagai *expression,* beberapa pengecualian yang dimaksud ditunjukkan melalui gambar 2.



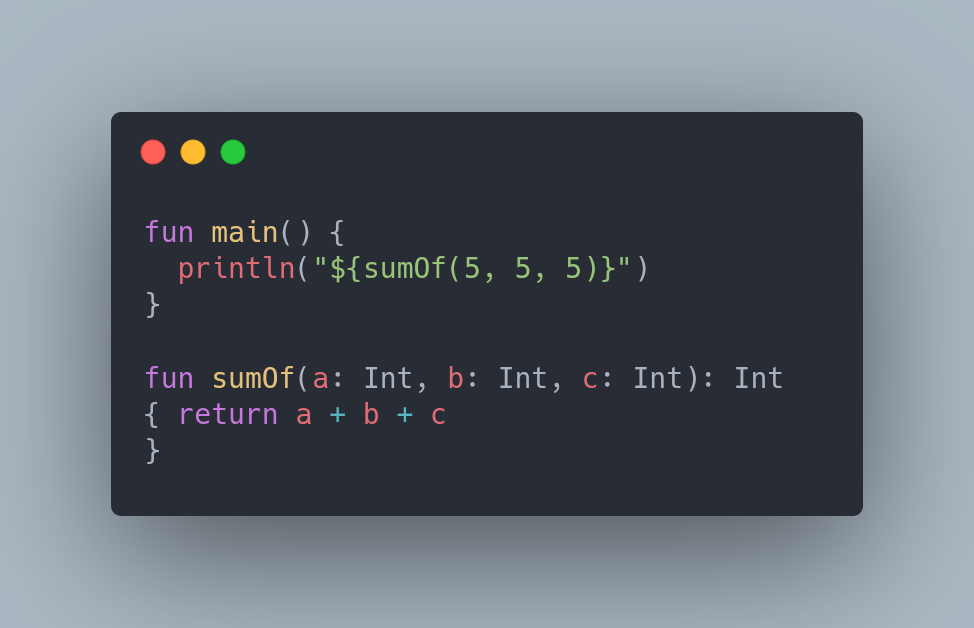
Gambar 2. Pengecualian expressions dalam Kotlin

Sedangkan *statements* adalah bagian terkecil dalam suatu program yang menyatakan suatu aksi yang dapat terdiri dari beberapa *expression.* Suatu *standalone expression* yang terdiri dari kombinasi antara *function* dan *value* juga termasuk sebuah *statement,* contohnya adalah updateUser(user).Pada gambar 3, ditunjukkan sepotong *codes* yang terdiri dari banyak *expressions* (contoh bestUser.name dan bestUser.surname) dalam 2 buah *statement*, dimana *statement 1* berfungsi untuk mendapatkan value dari bestUser dan statement 2 untuk menampilkan nama dari bestUser yang bersangkutan.



Gambar 3. Contoh statements dalam Kotlin

Kotlin memiliki ciri khas berupa tidak memiliki *statement separator* khusus layaknya *semicolon* atau titik koma (simbol ;) pada bahasa turunan C dan PHP, selama suatu *line codes* tidak memiliki awal atau akhiran berupa operator aktif maka dianggap sebagai satu kesatuan *statement* (satu baris) meskipun berupa *multi-line codes.* Meskipun begitu, *semicolon* pada Kotlin sebagai *statement separator* masih dapat digunakan jika ada beberapa *statement* pada baris yang sama hanya saja jarang digunakan. Agar *programmer* lebih mudah memahami *multi-line codes* sebagai sebuah baris, maka *line* kedua hingga *line* terakhir dari *statement* tersebut di awali dengan sebuah *tab.* Awal proses komputasi program berbasis Kotlin berada pada fungsi *entry poin* fun main(), sehingga semua *statements* yang ingin dipanggil harus tercantum pada fun main() tersebut. Pada gambar 4 ditunjukkan contoh *program line* sederhana dalam Kotlin, dimana fun sumOf() akan dipanggil pada fun main()*.*



hasil:



Gambar 4. Program sederhana untuk menghitung jumlah 3 integer dalam Kotlin

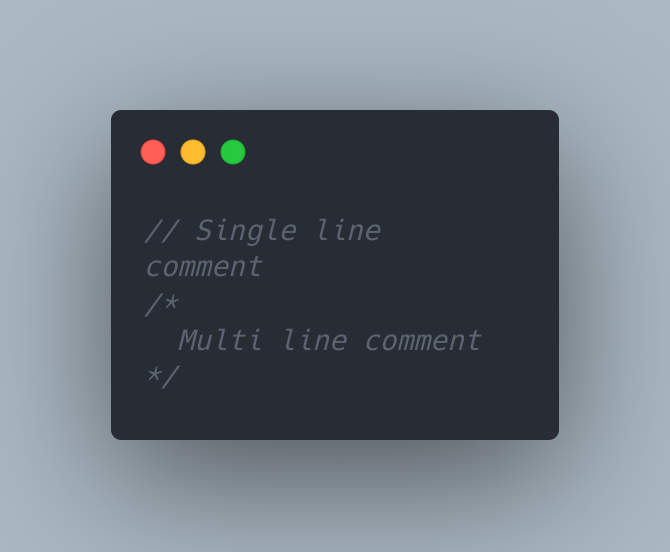
Pada bahasa pemograman, *keyword* merupakan suatu kata yang sudah tersimpan dalam bahasa pemograman dan tidak dapat digunakan oleh *user* sebagai *identifier,* baik *variables, functions,* maupun *labels.* Pada Kotlin, beberapa *keywords* yang umum digunakan di antaranya :

1. fun, digunakan untuk mendeklarasikan *function.*
2. var, digunakan untuk mendeklarasikan variabel yang nilainya dapat berubah (*mutable variable*).
3. val, digunakan untuk mendeklarasikan variabel yang nilainya tidak dapat diubah (*immutable variable*).
4. if, digunakan untuk memulai *if expression* (percabangan sesuai syarat).
5. else, digunakan untuk mendefinisikan percabangan yang bernilai FALSE.

*Identifier* adalah suatu kata yang dapat digunakan oleh *user* untuk mendefinisikan sebuah *variables, functions,* maupun *labels.* Seperti pada penjelasan sebelumnya, kata-kata yang termasuk dalam *keywords* tidak dapat digunakan sebagai *identifier.* Selain itu, beberapa syarat dalam menentukan kata *identifier* adalah sebagai berikut :

1. Disarankan menggunakan kata yang memiliki arti agar mudah dipahami.
2. Dapat diawali dengan *underline* ( \_ ).
3. Tidak boleh memiliki *whitespace,* seperti *space*, *new line* (*enter*), maupun *tab*.
4. Bersifat *case sensitive,* huruf kapital dan huruf kecil dianggap sebagai karakter yang berbeda.
5. Tidak boleh memiliki karakter simbol &, !, @, $, #, dll.
6. Jika terdiri lebih dari satu kata, kata kedua dan seterusnya diawali dengan huruf kapital, contohnya updateStatus dan sumOfTwo .
7. Disaranan menggunakan bahasa Inggris, dimana dapat membedakan data tunggal atau data jamak hanya dengan menambahkan huruf “..-s” atau “…-SSSes”

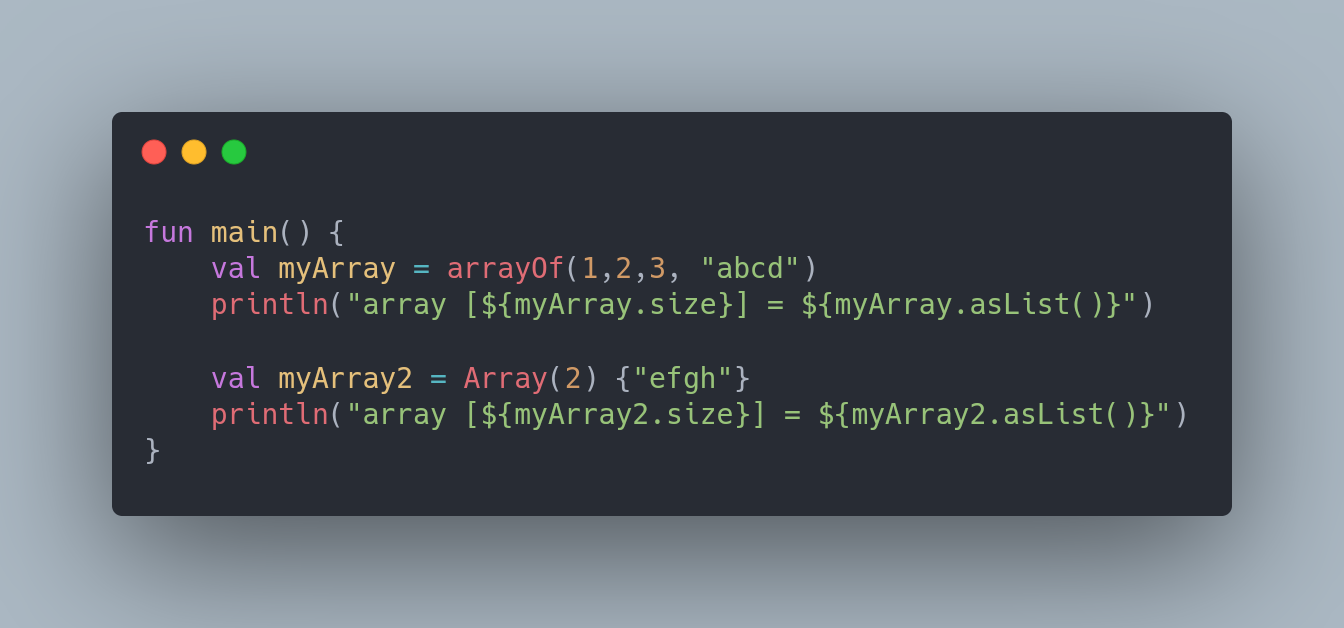
Sama seperti bahasa pemograman lainnya, Kotlin menyediakan *comment syntax* untuk memberikan catatan yang tidak akan memengaruhi *source code* program saat dijalankan. *Comment* dapat digunakan sebagai penjelasan dari *code* yang bersangkutan dan berfungsi sebagai dokumentasi agar lebih mudah dalam memahami algoritmanya. *Comment syntax* dalam Kotlin dapat berupa *single line* maupun *multi-line* seperti pada gambar 5.



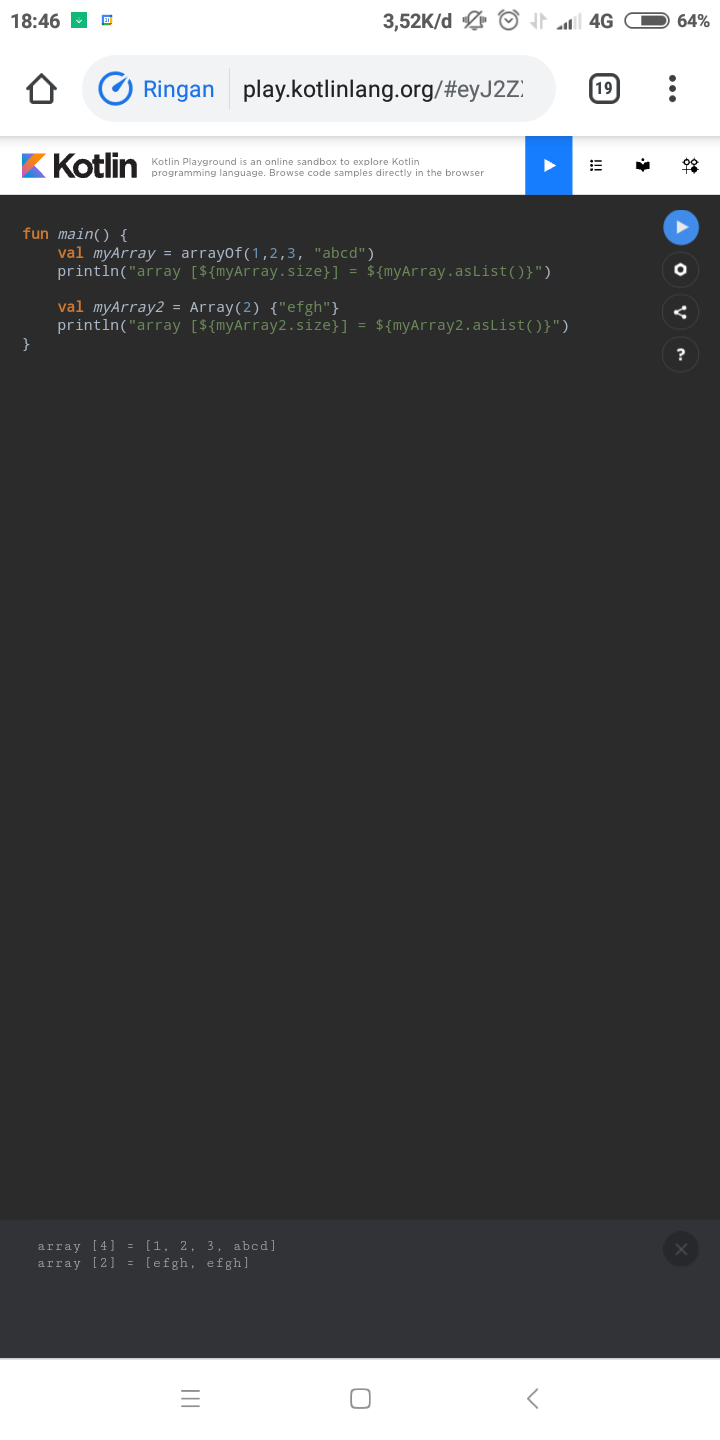
Gambar 5. Comment syntax pada Kotlin

Kotlin merupakan *object oriented programming* (OOP) yang memungkinkan untuk memecahkan permasalahan berbasis objek, artinya seluruh *syntax* pada Kotlin dapat memanggil *method* dan *property* lain. Namun beberapa tipe data yang tidak dapat memanggil *property* maupun *method* yang disebut sebagai *data primitive* yang hanya memiliki sebuah *value*, seperti *number, character, string,* dan *array*.

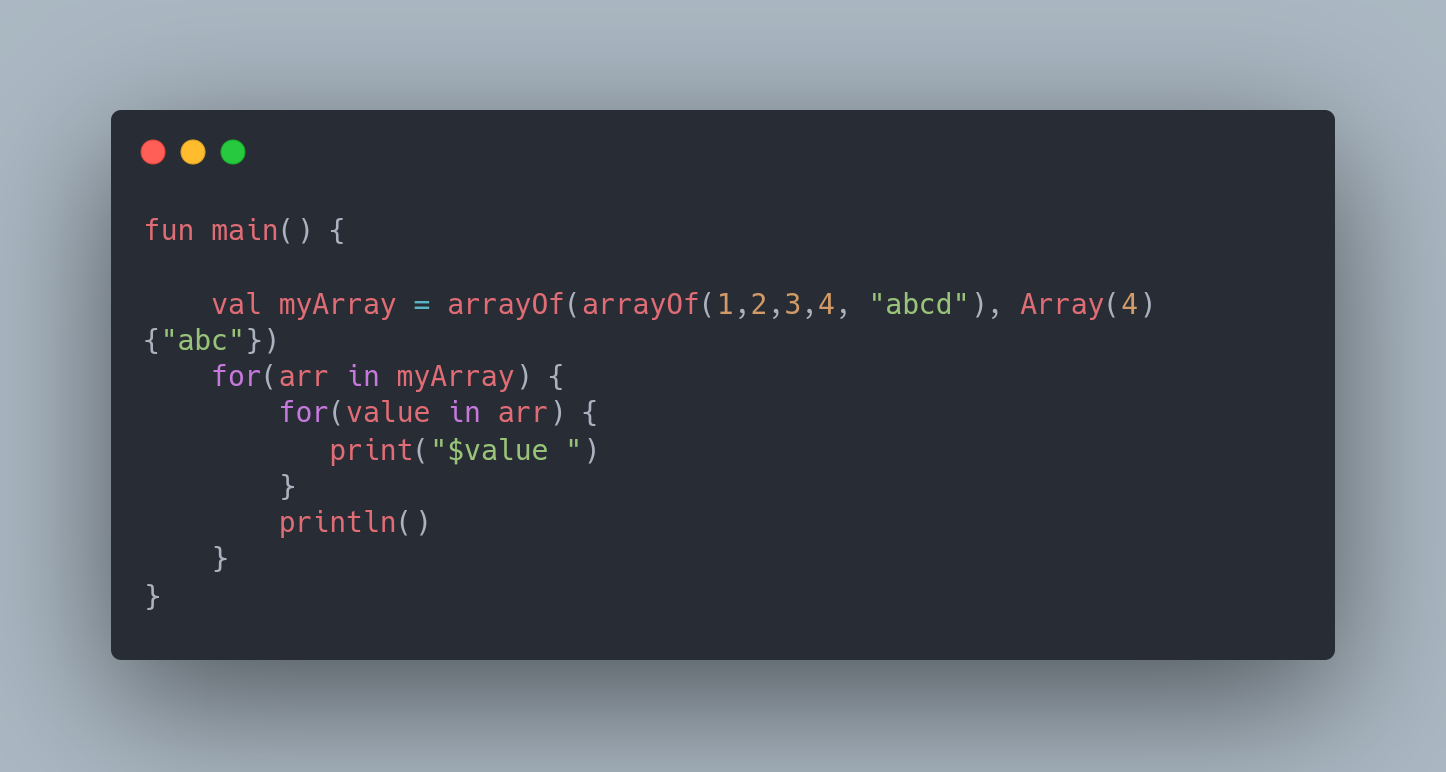
1. *Number*, berupa angka yang terdiri dari *integer* berupa bilangan bulatdan *floating-point* berupa *float* maupun *double*
2. *Boolean,* berupa nilai logika berupa *true* dan *false*. Digunakan dalam perbandingan logika menggunakan operator AND (&&), OR(||), maupun NOT atau negasi (!).
3. *Character,* berupa sebuah karakter dimana khusus untuk *special character* dengan spesifikasi khusus membutuhkan sebuah *escaping backslash* sebelumnya ( / ) atau *sequence syntax* untuk *unicode* xuFF000.
4. *String,* berupa sebuah kata atau kalimat yang membutuhkan *double quotes* ( “ ) untuk meyimpan sebagai string bukan *identifier,* atau dapat menggunakan *triple quotes* untuk *multi-line string* sebelum *string* yang bersangkutan (“””))*.*
5. *Array*, berupa variabel yang memiliki lebih dari sebuah nilai menggunakan syntax arrayOf(<value>) maupun Array(<size>){<value>} seperti pada gambar 6. Untuk *multi-dimensional array* juga menggunakan *syntax* sebelumnya, hanya saja value-nya juga berbentuk *array* sehingga menjadi arrayOf(arrayOf(<>), arrayOf(<>)) seperti pada gambar 7.



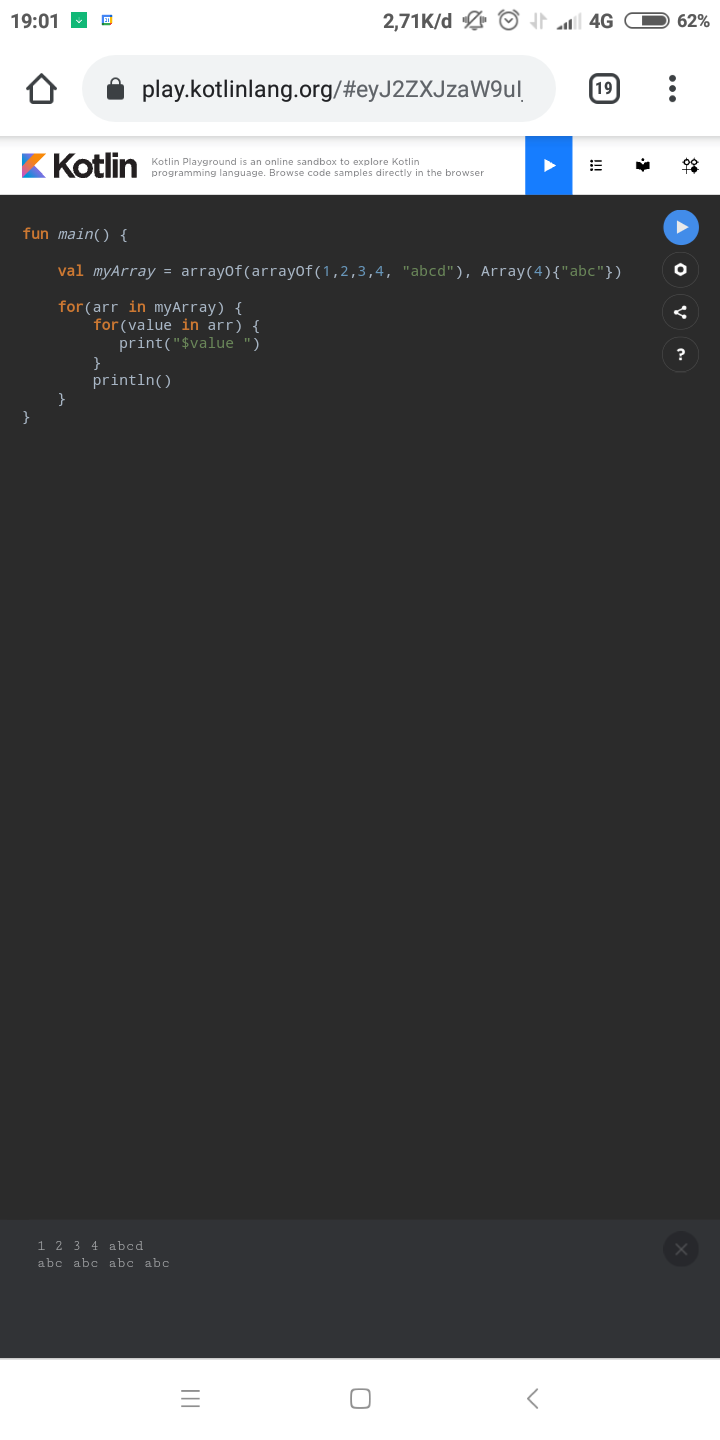
hasil:



Gambar 6. Pembuatan array pada Kotlin

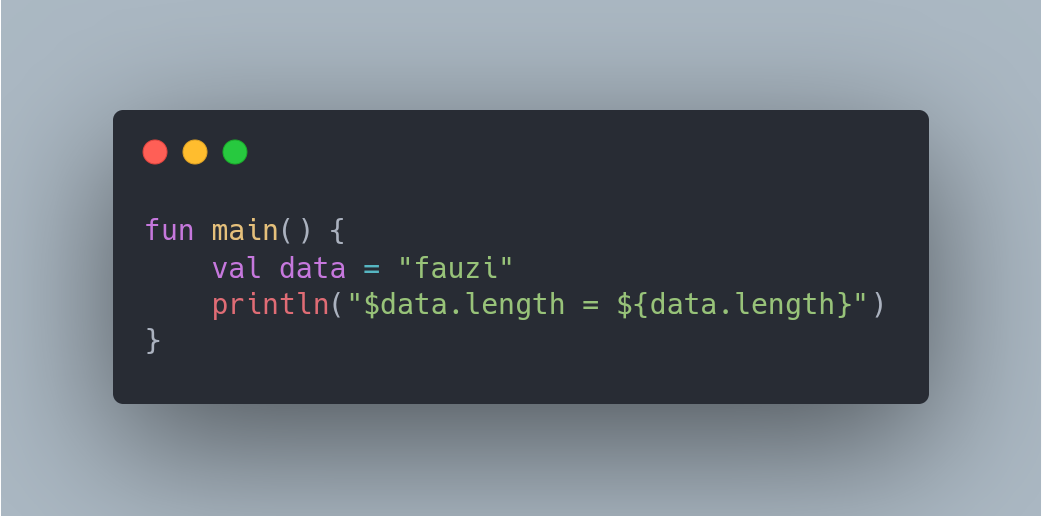


hasil:

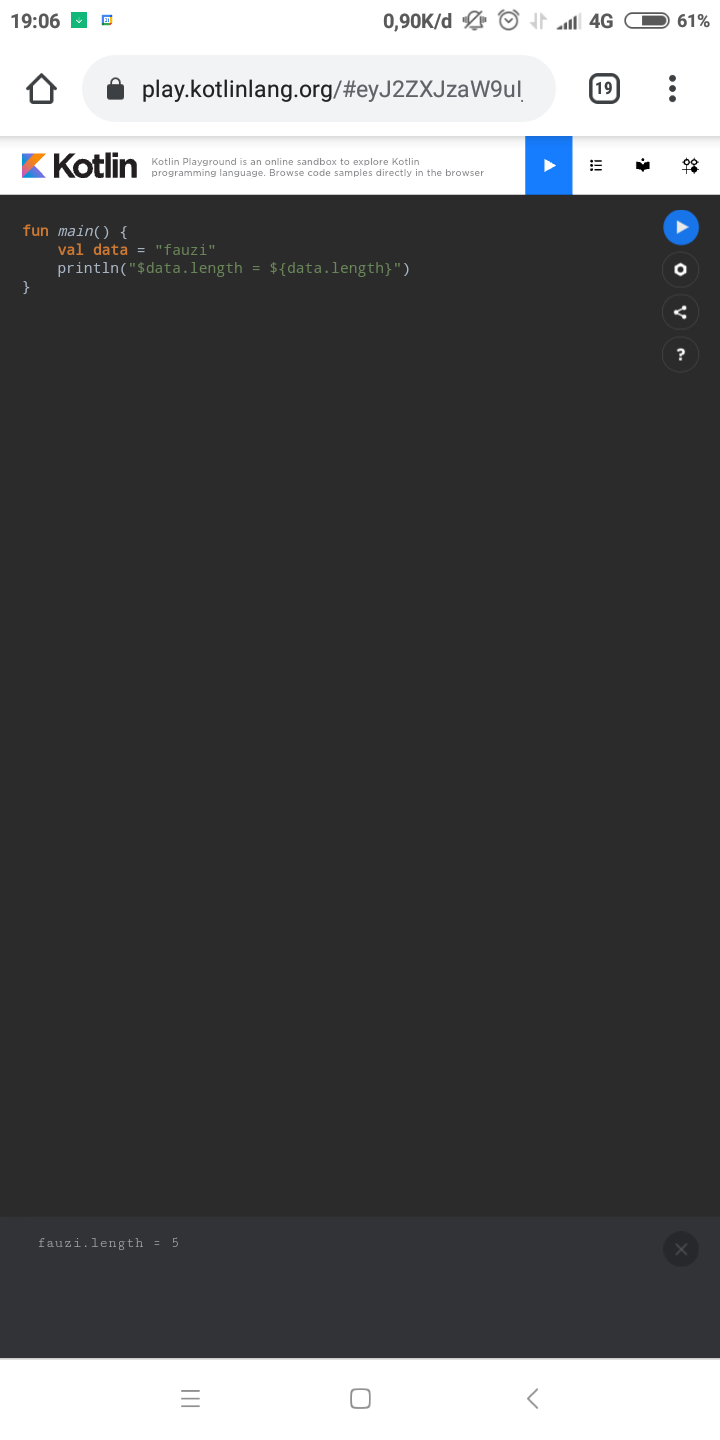


Gambar 7. Pembuatan multi-dimensional array pada Kotlin

Karakteristik dan penggunaan detail dari *data primitive* pada Kotlin tidak jauh berbeda dengan bahasa pemograman lainnya. Untuk menggunakan *value* dari *data primitive*, dapat memanfaatkan symbol *dollar* ($) dan *curly braces* ({}) seperti pada gambar 8. Untuk mengecek tipe data suatu variabel, Kotlin menyediakan *type checking* menggunakan is *operator* seperti pada gambar 9 dimana jika sesuai maka menghasilkan nilai *true.*



hasil:



Gambar 8. Penggunaan simbol dollar ($) untuk data primitive pada Kotlin



hasil:



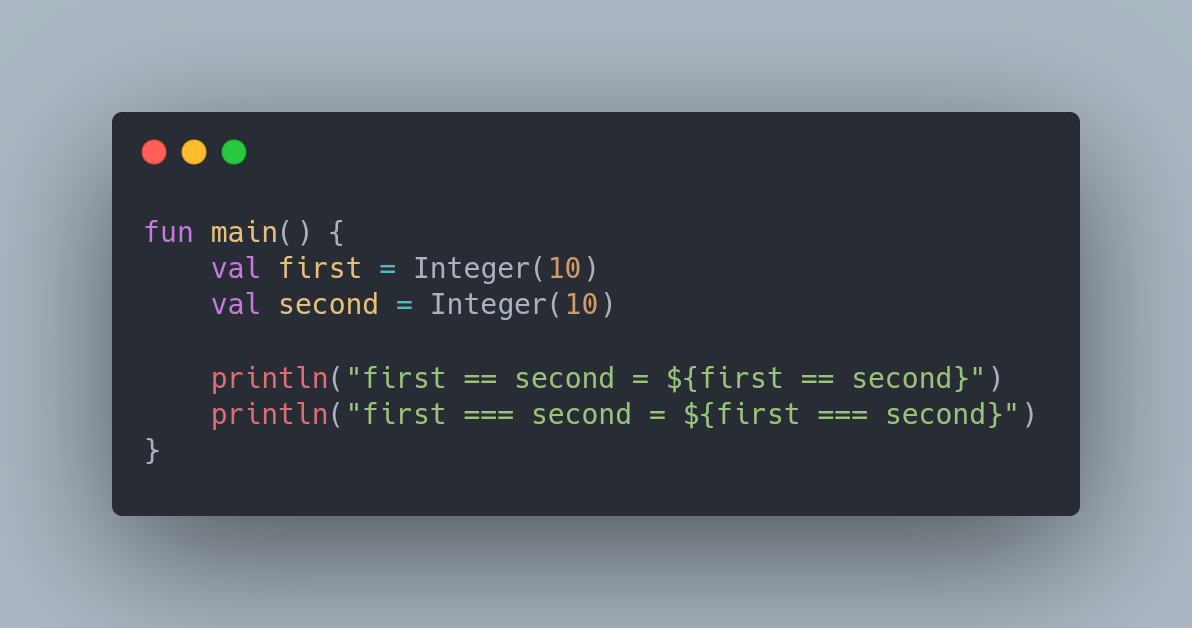
Gambar 9. Type check dengan is operator pada Kotlin

Pembuatan variabel pada Kotlin memiliki 2 jenis, yaitu *immutable variable* dan *mutable variable* menggunakan syntax val dan var seperti pada penjelasan pada bagian *Keywords* Kotlin sebelumnya. Penggunaannya dapat menyebutkan tipe data secara eksplisit maupun tidak seperti pada gambar 10, jika tidak diberikan maka tipe data suatu variabel mengikuti *value* yang disimpan pada variabel yang bersangkutan.



Gambar 10. Deklarasi variabel pada Kotlin

Secara umum operator pada Kotlin selayaknya pada bahasa pemograman lain, memiliki *matemathic operator* seperti +, -, /, \*, maupun %, serta *logical operator* seperti &&, ||, !. Hanya saja, *comparasion operator* pada Kotlin memiliki sedikit perbedaan dengan bahasa pemograman lainnya. Di Kotlin, terdapat *equality operator* dan *referential equality operator* yang berbeda karena adanya perbedaan objek atau nilai yang dibandingkan. *Equality operator* dengan syntax “==” dan “!=” membandingkan nilai dari kedua variabel, sedangkan *referential equality operator* membandingkan apakah kedua variabel merujuk pada satu objek yang sama atau tidak, perbedaan keduanya terlihat melalui gambar 11.



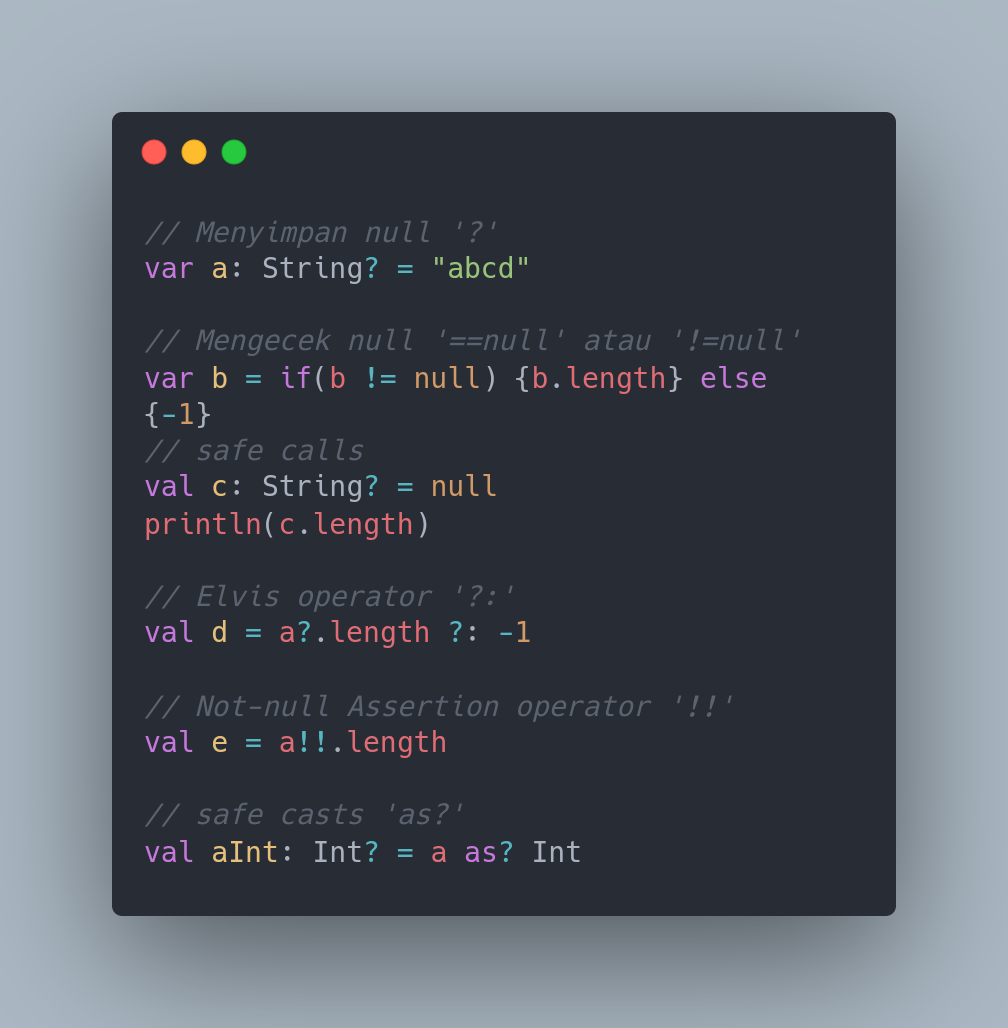
hasil:



Gambar 11. Perbedaan comparison operator pada Kotlin

Hal lain yang membedakan Kotlin dengan bahasa pemograman lainnya adalah Kotlin menyediakan fitur untuk menghindari adanya nilai null pada suatu variabel yang disebut dengan *Null Safety,* dimana menghindari adanya nilai NullPointerException atau NPE pada Java *.* Ada beberapa variabel yang bisa memuat *null* (*nullable references*) ataupun tidak bisa bernilai *null* (*non-null references*). Seperti contoh, tipe data String tidak dapat bernilai *null* sehingga saat menyimpan nilai tersebut, code tidak dapat di *compile.* Operator yang berkaitan dengan mekanisme *non-null* dan *null type* di antaranya :

1. Membuat suatu variabel agar dapat menyimpan
2. Mengecek suatu variabel apakah null atau tidak
3. Membuat suatu kode hanya dapat dieksekusi jika suatu variabel tidak bernilai null. Akan menghasilkan null jika suatu variabel bernilai null dan bernilai *true* jika non-null (*Safe calls*)
4. *Safe calls* yang nilai *non-null* nya bisa didefinisikan (*Elvis Operator*)
5. Memaksa suatu nilai menjadi *non-null type* (*Not null assertion operator*)
6. Menghindari adanya ClassCastException (*safe casts*)



Gambar 12. Operator untuk null dan non-null type