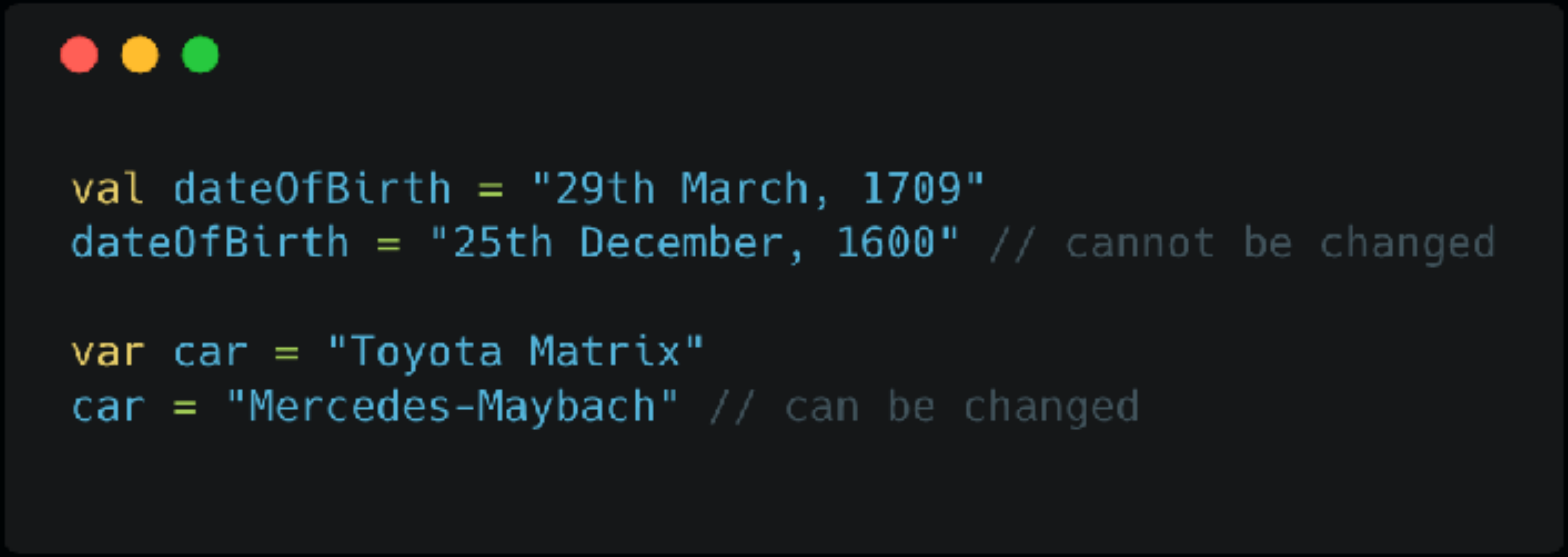
Kotlin Journey-I

Declaring variables

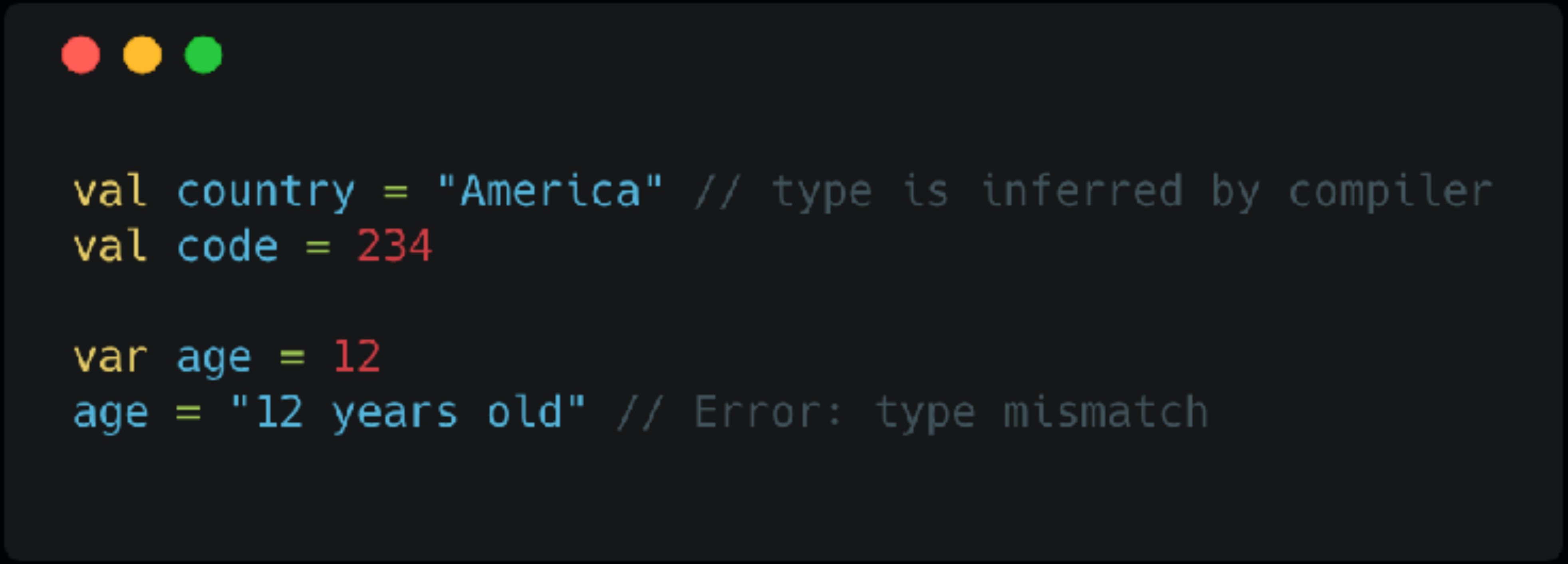
- Variable declaration using val and var keywords.
- val: To declare constants and make variables immutable
- var: To declare mutable variables



```
val dateOfBirth = "29th March, 1709"
dateOfBirth = "25th December, 1600" // cannot be changed

var car = "Toyota Matrix"
car = "Mercedes-Maybach" // can be changed
```

Type Inferencing

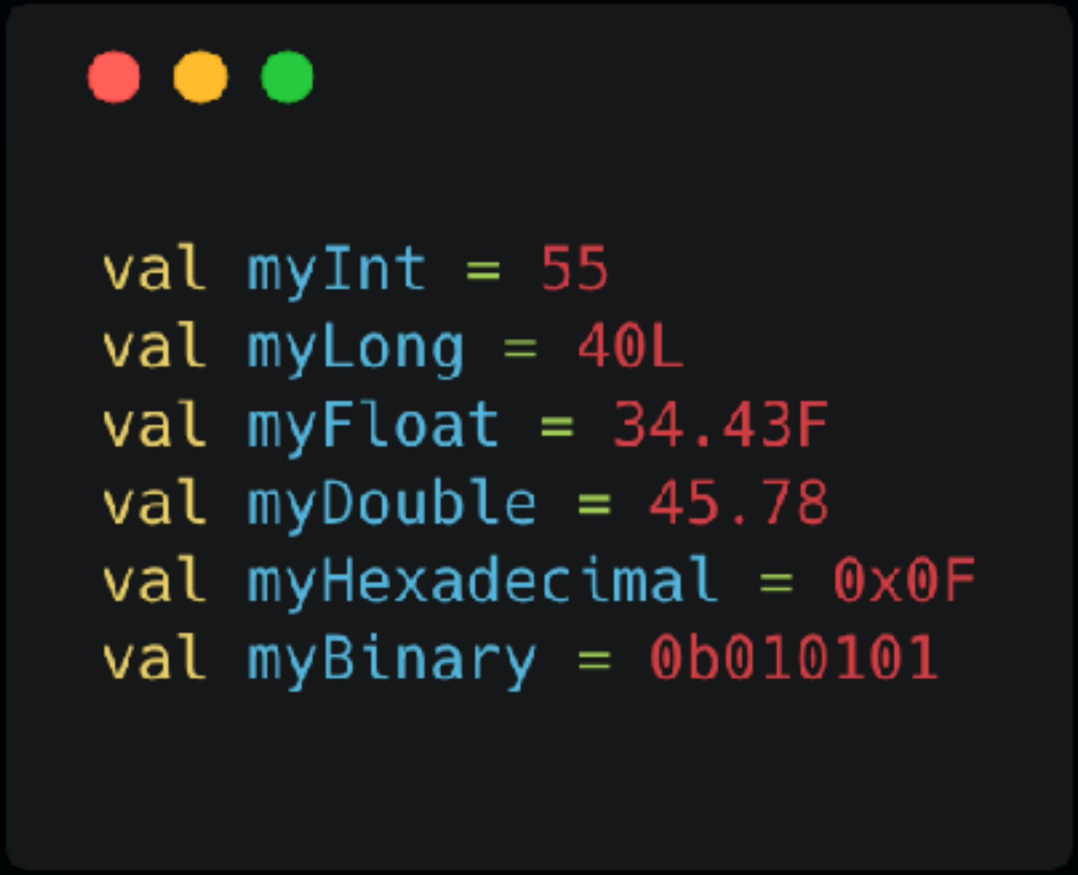
- Mechanism to find out type from the context of the variable.



```
val country = "America" // type is inferred by compiler
val code = 234

var age = 12
age = "12 years old" // Error: type mismatch
```

Data Type

- Long
- Int
- Short
- Byte
- Double
- Float
- Boolean



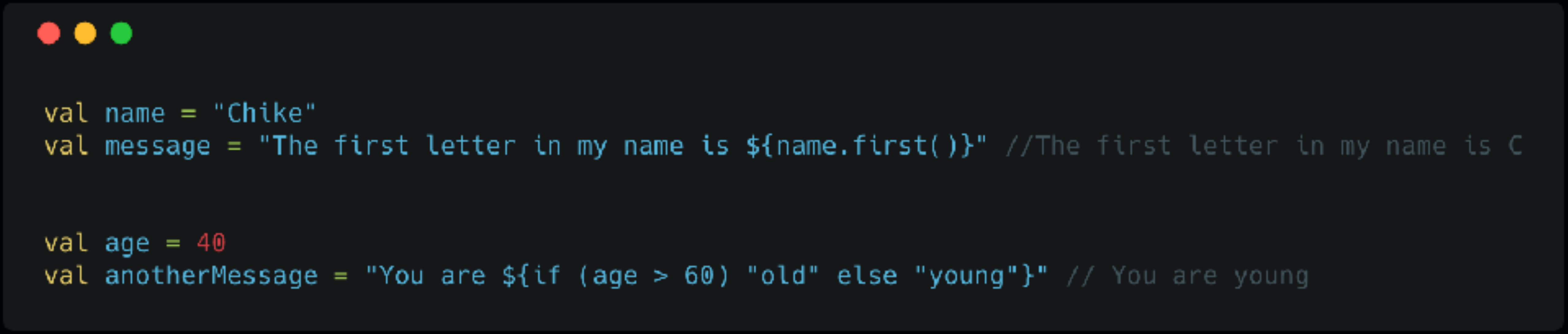
```
val myInt = 55
val myLong = 40L
val myFloat = 34.43F
val myDouble = 45.78
val myHexadecimal = 0x0F
val myBinary = 0b010101
```

Interconvertible to each other by following:

1. **toByte( )**
2. **toInt( )**
3. **toLong( )**
4. **toFloat( )**
5. **toChar( )**
6. **toDouble( )**

Strings

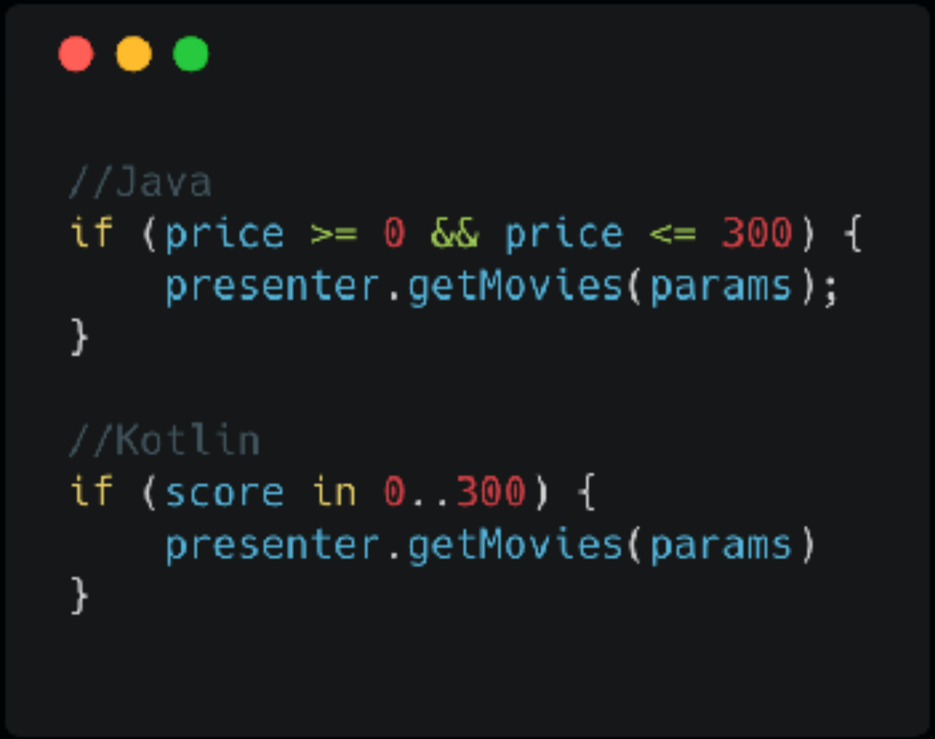
- Simple creation of String with val or var.
- Supports String interpolation and create dynamic String
- You can call methods from the interpolated Strings.



```
val name = "Chike"
val message = "The first letter in my name is ${name.first()}" //The first letter in my name is C


val age = 40
val anotherMessage = "You are ${if (age > 60) "old" else "young"}" // You are young
```

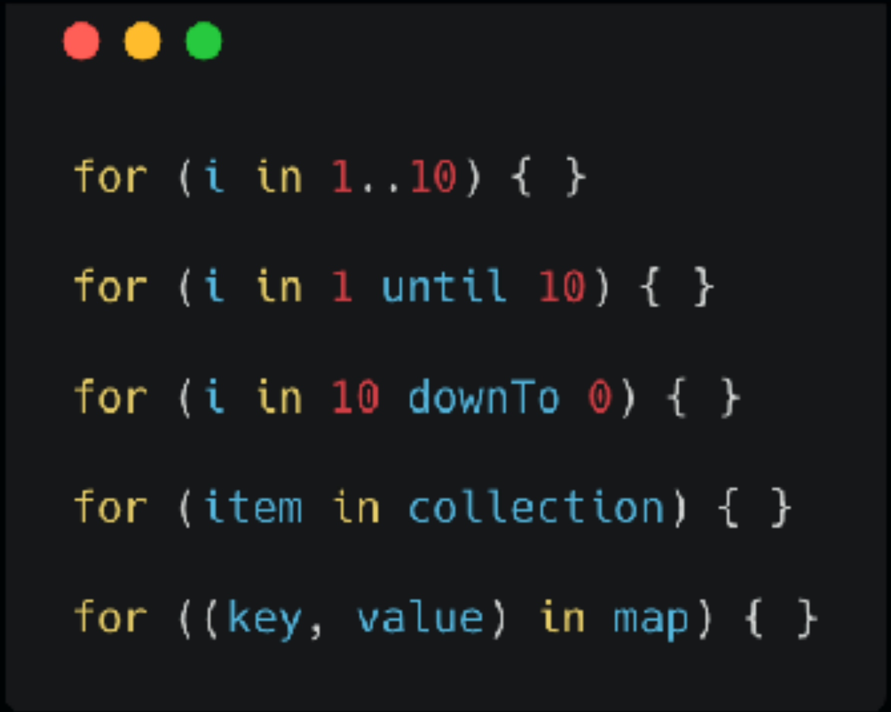
Flow Control

- **If conditions:**



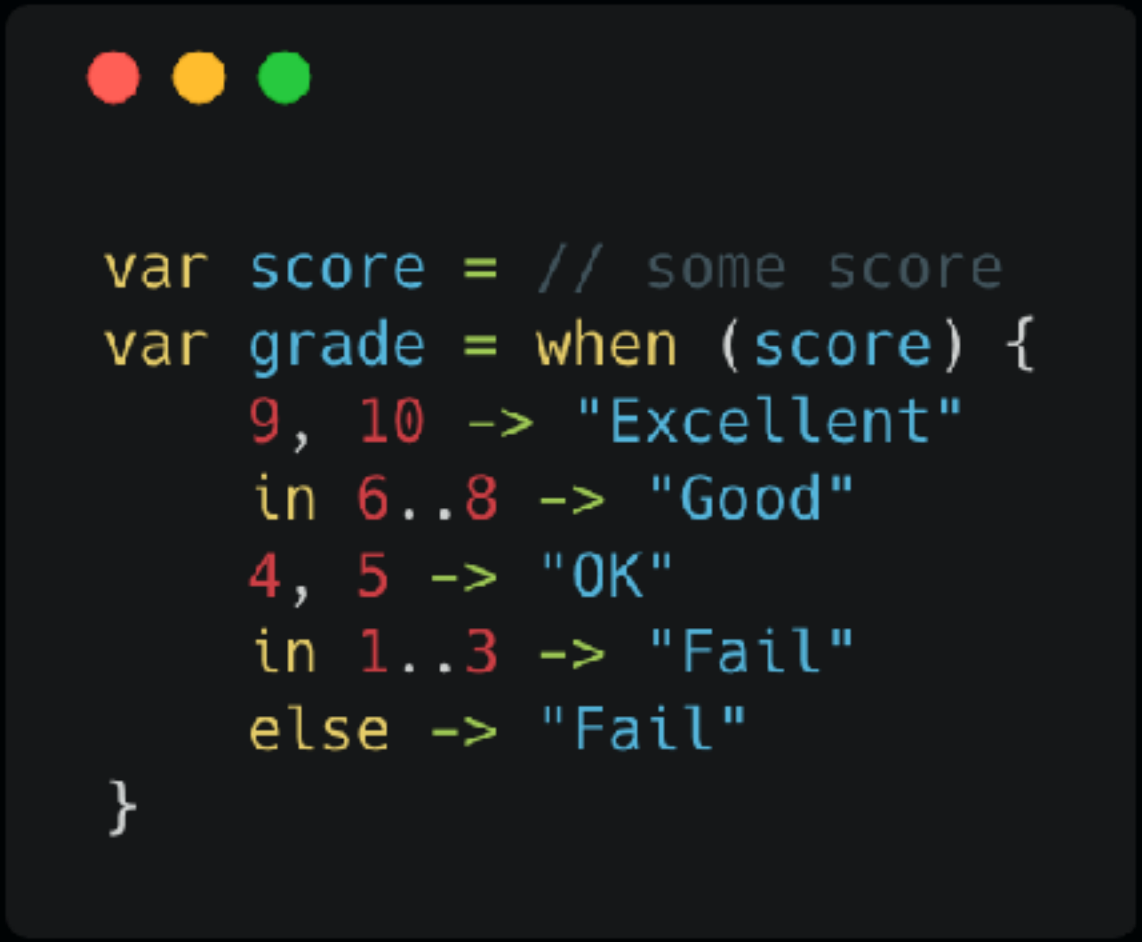
```
//Java
if (price >= 0 && price <= 300) {
    presenter.getMovies(params);
}

//Kotlin
if (score in 0..300) {
    presenter.getMovies(params)
}
```

- **For loops:**



```
for (i in 1..10) { }

for (i in 1 until 10) { }

for (i in 10 downTo 0) { }

for (item in collection) { }

for ((key, value) in map) { }
```

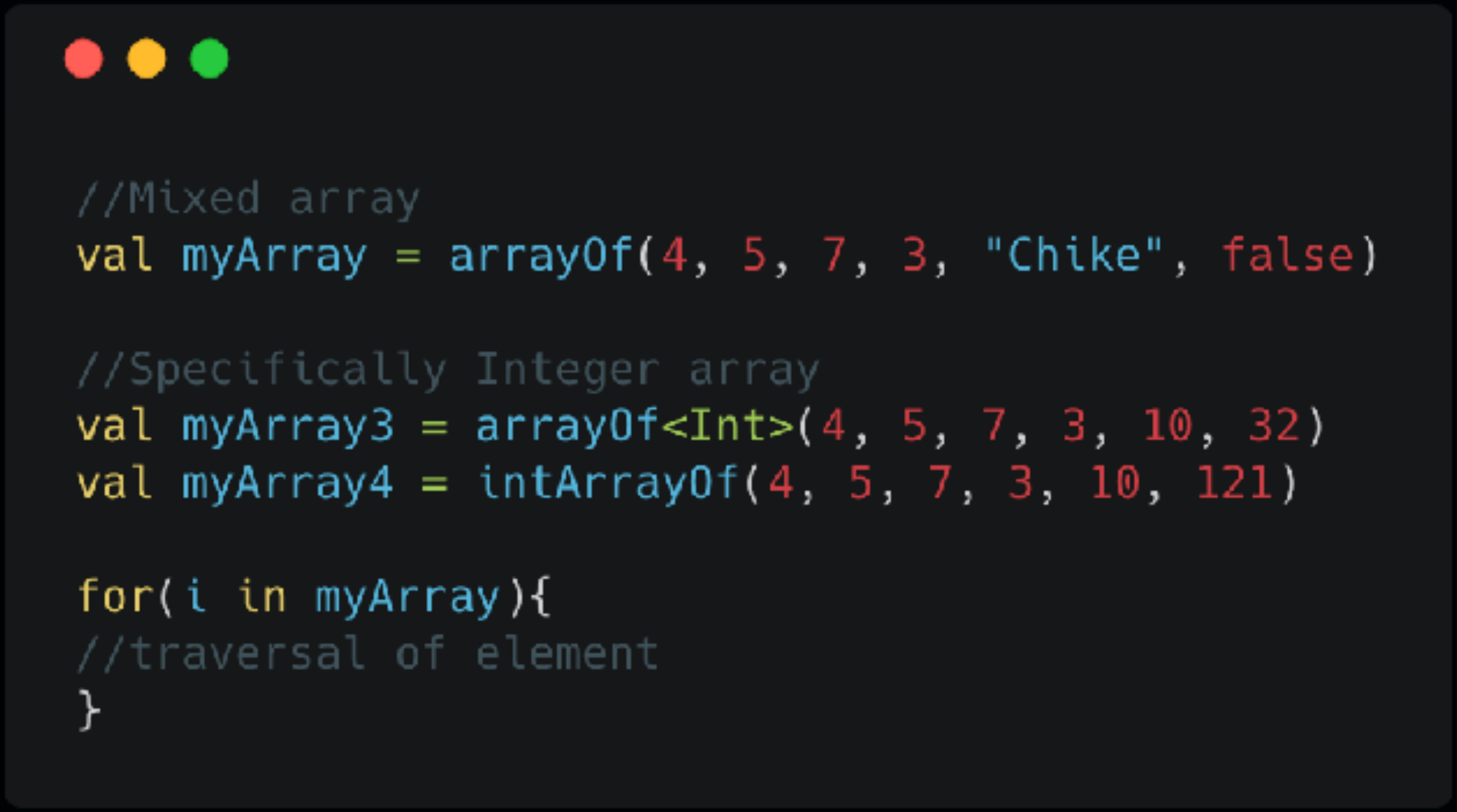
- **Switch Cases**



```
var score = // some score
var grade = when (score) {
    9, 10 -> "Excellent"
    in 6..8 -> "Good"
    4, 5 -> "OK"
    in 1..3 -> "Fail"
    else -> "Fail"
}
```

Arrays

- arrayOf( ) to declare array with mixed number of elements
- Specific array by: arrayOf<Int>(…….) or intArrayOf(…..)



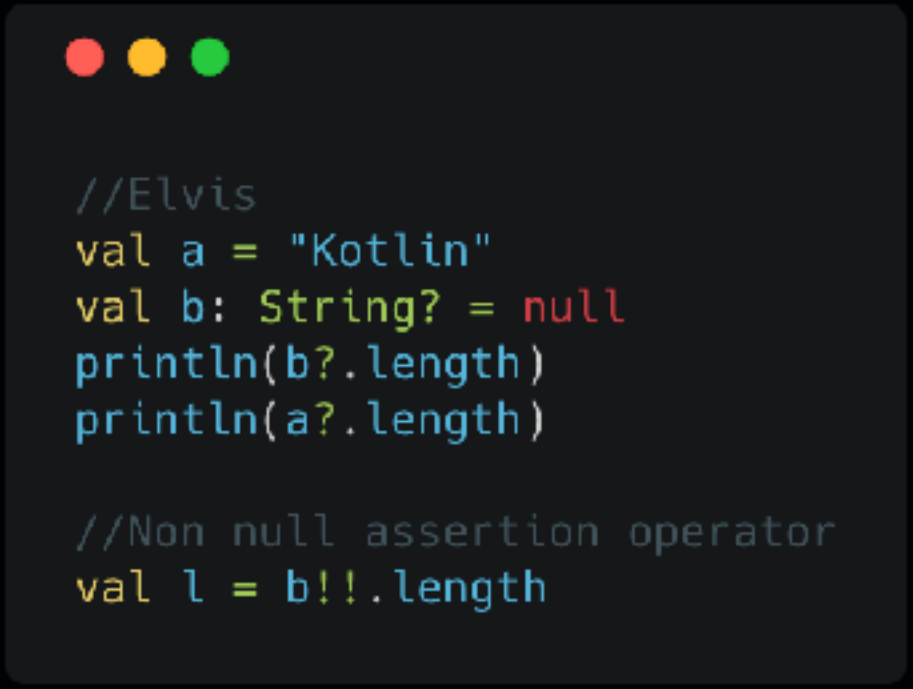
```
//Mixed array
val myArray = arrayOf(4, 5, 7, 3, "Chike", false)

//Specifically Integer array
val myArray3 = arrayOf<Int>(4, 5, 7, 3, 10, 32)
val myArray4 = intArrayOf(4, 5, 7, 3, 10, 121)

for(i in myArray){
//traversal of element
}
```

Operators

- **as** for type casting
- **in** during iteration over collection or mentioning some range.
- **is** for checking for certain type
- Structural equality by == and negated counterpart != .
- Referential equality === and counterpart !== .
- Elvis operator: ? Safe calls

  Here you are doing safe calls no NPE
- Non null assertion operators: !!

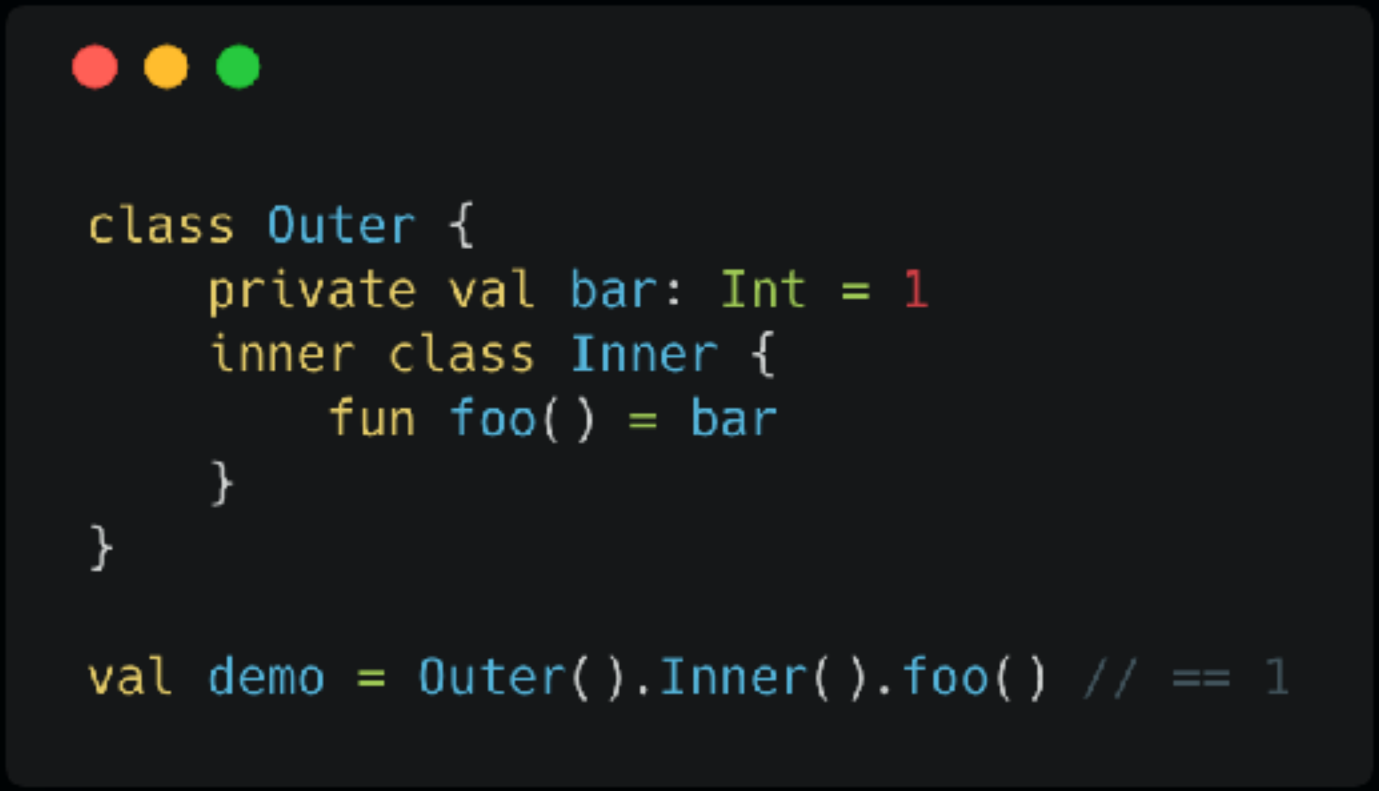
  Here you will ask for NPE if anything null comes up.
- **$ String template**



```
//Elvis
val a = "Kotlin"
val b: String? = null
println(b?.length)
println(a?.length)

//Non null assertion operator
val l = b!!.length
```

Keywords

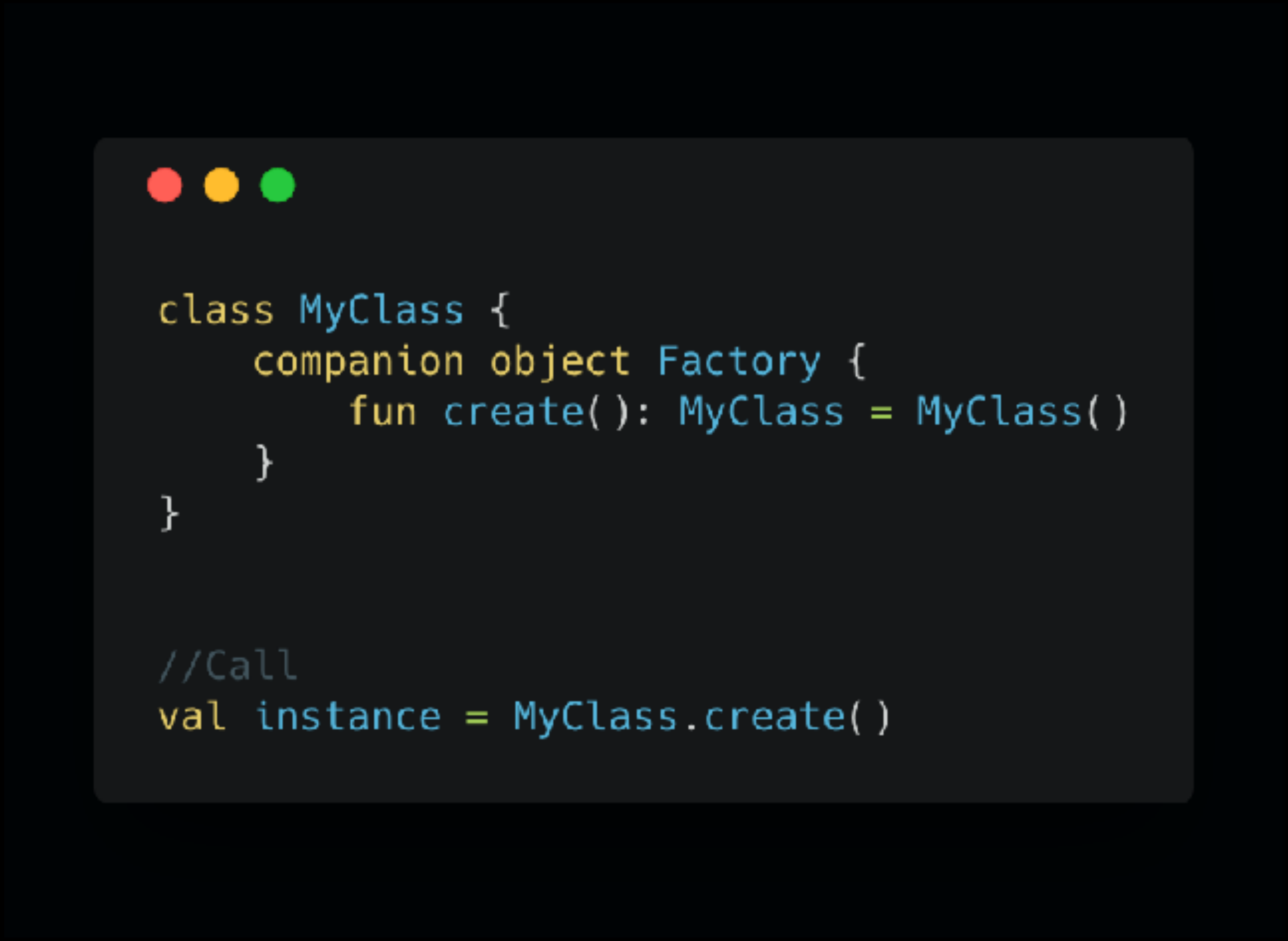
- **inner**: A class may be marked as “inner” to be able to access members of outer class. Inner classes carry a reference to an object of an outer class:



```kotlin
class Outer {
    private val bar: Int = 1
    inner class Inner {
        fun foo() = bar
    }
}

val demo = Outer().Inner().foo() // == 1
```

- **internal:** Extra visibility available in Kotlin. If you mark internal it is visible throughout in same module.
- **open:** Classes in Kotlin by default inherits from Any type. To declare explicitly Parent we keep the parent class open. Opposite to final. By default all are final.

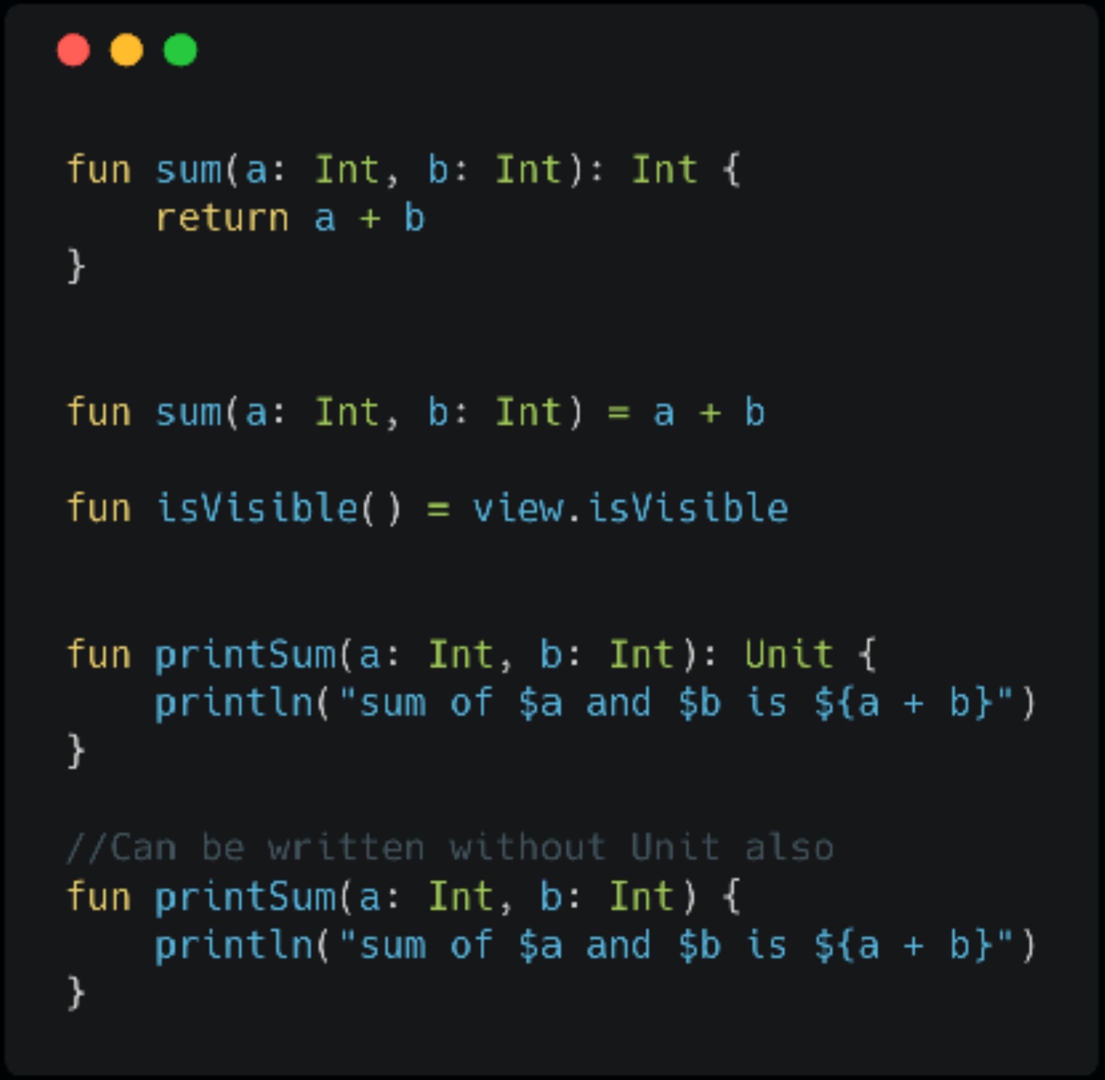
- Companion object:



```kotlin
class MyClass {
    companion object Factory {
        fun create(): MyClass = MyClass()
    }
}


//Call
val instance = MyClass.create()
```

Function

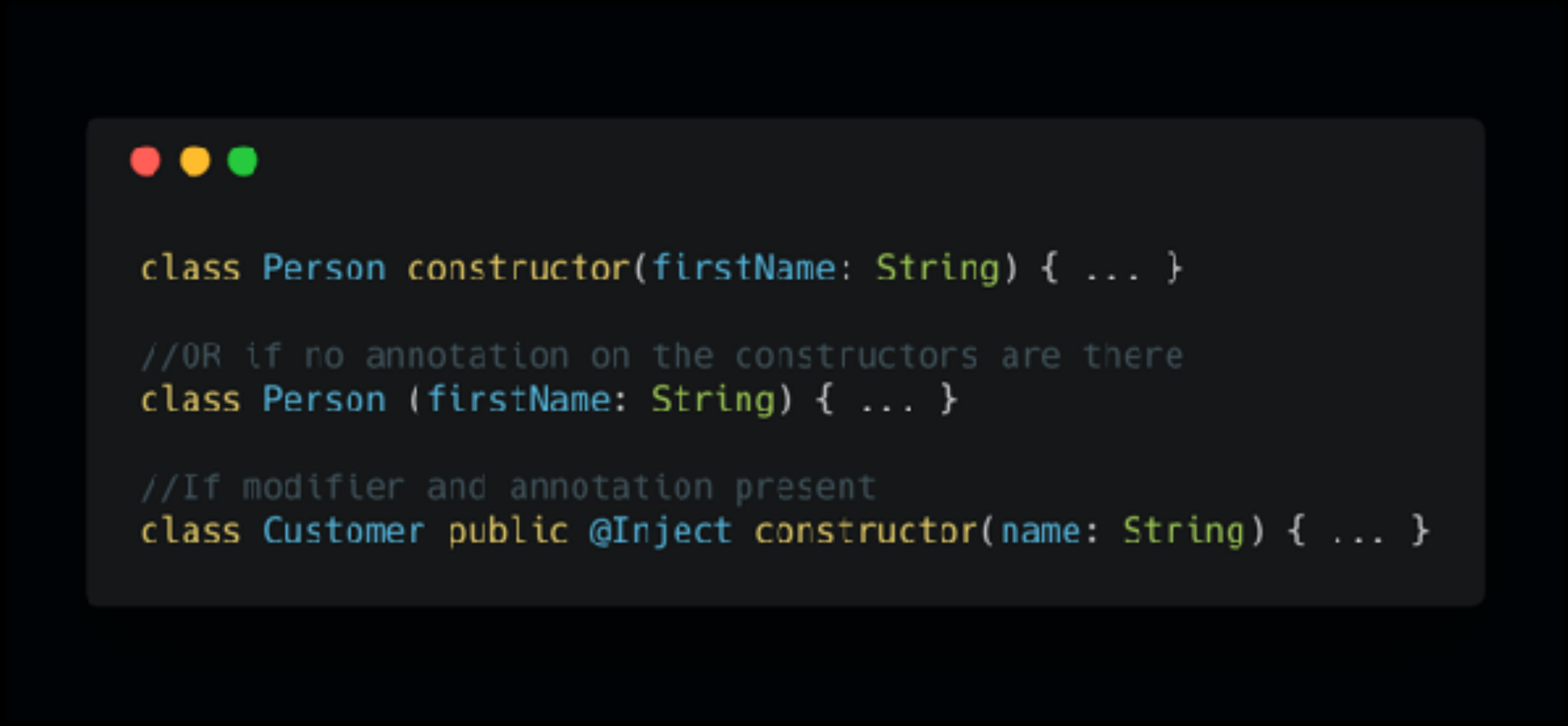
1. Normal function with arguments as integers and return integer.
2. Function with an expression body and inferred return type.
3. Function returning Unit void equivalent to void.
4. Can be written without Unit also.



```kotlin
fun sum(a: Int, b: Int): Int {
    return a + b
}


fun sum(a: Int, b: Int) = a + b

fun isVisible() = view.isVisible


fun printSum(a: Int, b: Int): Unit {
    println("sum of $a and $b is ${a + b}")
}

//Can be written without Unit also
fun printSum(a: Int, b: Int) {
    println("sum of $a and $b is ${a + b}")
}
```

Constructors

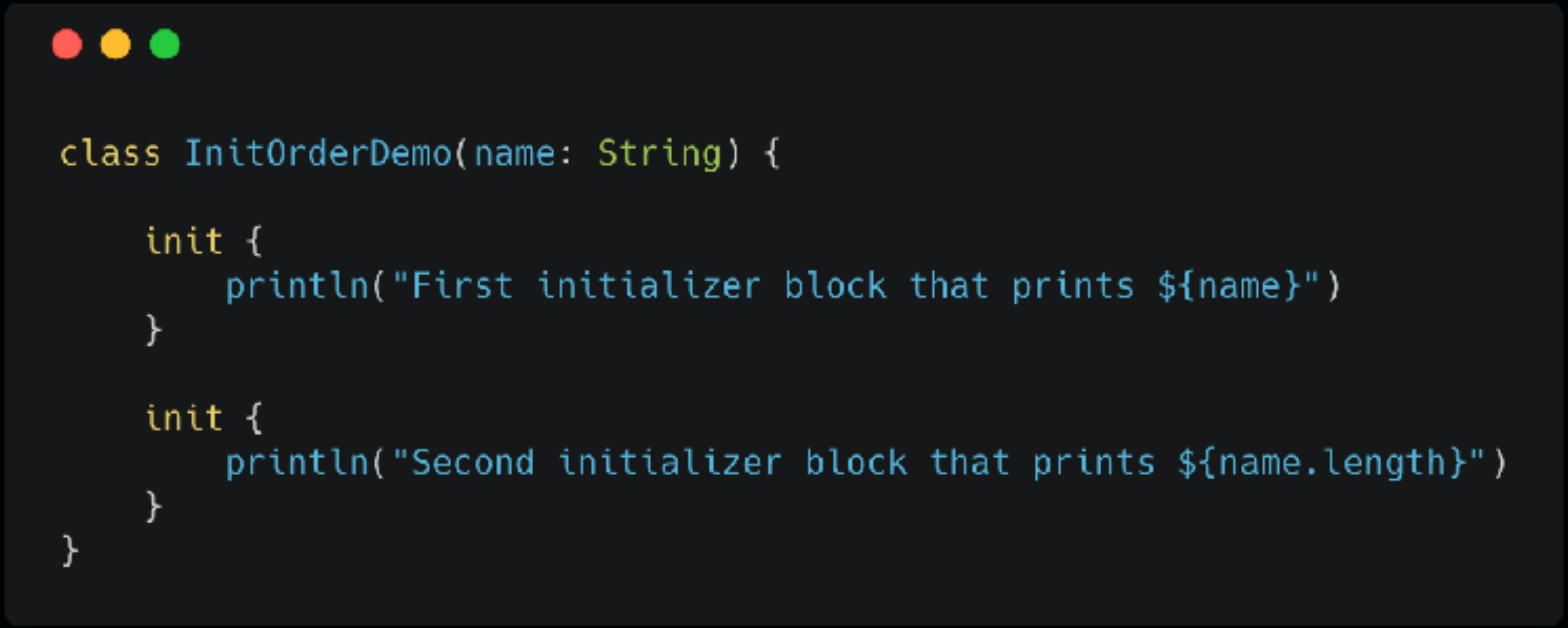
- Kotlin can have one primary constructor and one or more secondary constructors.



```
class Person constructor(firstName: String) { ... }

//OR if no annotation on the constructors are there
class Person (firstName: String) { ... }

//If modifier and annotation present
class Customer public @Inject constructor(name: String) { ... }
```

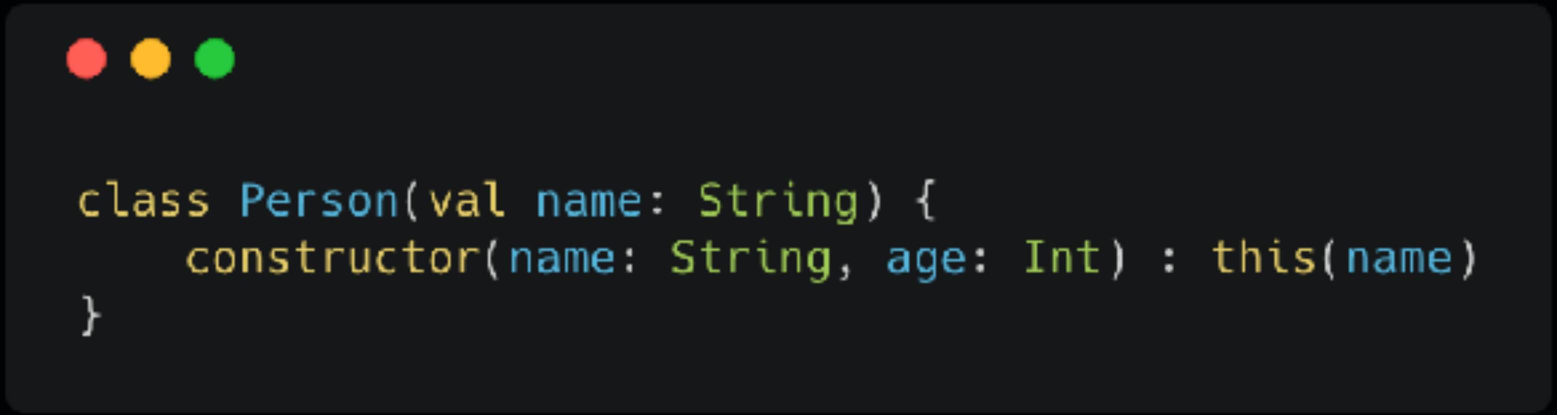
- For initializer code init blocks are there no code in this primary constructor is allowed. Initializer blocks are executed in same order they are written.

- These init are executed during instance initialization.



```
class InitOrderDemo(name: String) {

    init {
        println("First initializer block that prints ${name}")
    }

    init {
        println("Second initializer block that prints ${name.length}")
    }
}
```

Secondary Constructors:



```
class Person(val name: String) {
    constructor(name: String, age: Int) : this(name)
}
```

“If the class has a primary constructor, each secondary constructor needs to delegate to the primary constructor, either directly or indirectly through another secondary constructor(s). Delegation to another constructor of the same class is done using the this keyword”

Thank you