Introduction to Kotlin Workshop for Intermediate Android Developers

June 9 + 10, 2018

@velosmobile

Intros & Logistics

Introductions

Seetha Annamraju

Chris King

Logistics

- Feel free to tweet: @velosmobile / #velosweekend
- Velos staff
- Format of Workshop/ Slides
- Any questions?

Prerequisites

- Experience with Android Development in Java
- Android Studio 3.0 +
- EduTools plugin for Kotlin Koans or https://try.kotlinlang.org/
- New Android project setup

References

- http://kotlinlang.org/docs/reference/
- *Kotlin in Action*: Dmitry Jemerov & Svetlana Isakova

Introduction to Kotlin

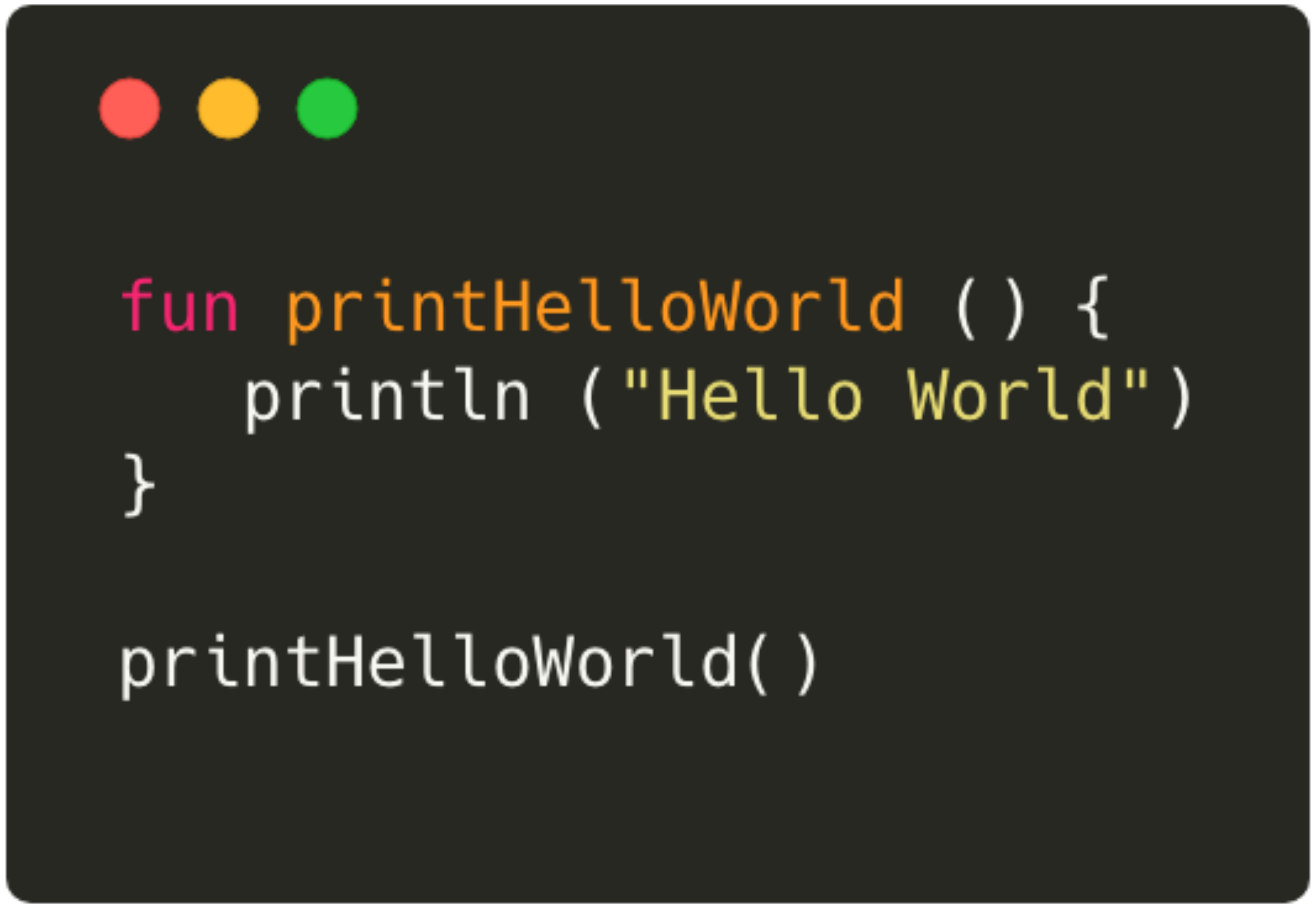
What is Kotlin?

- Why learn Kotlin?
- Provides “more concise, productive, safer alternative to Java”
- Made by developers
- It’s cool. It’s supported.

Kotlin vs. Java

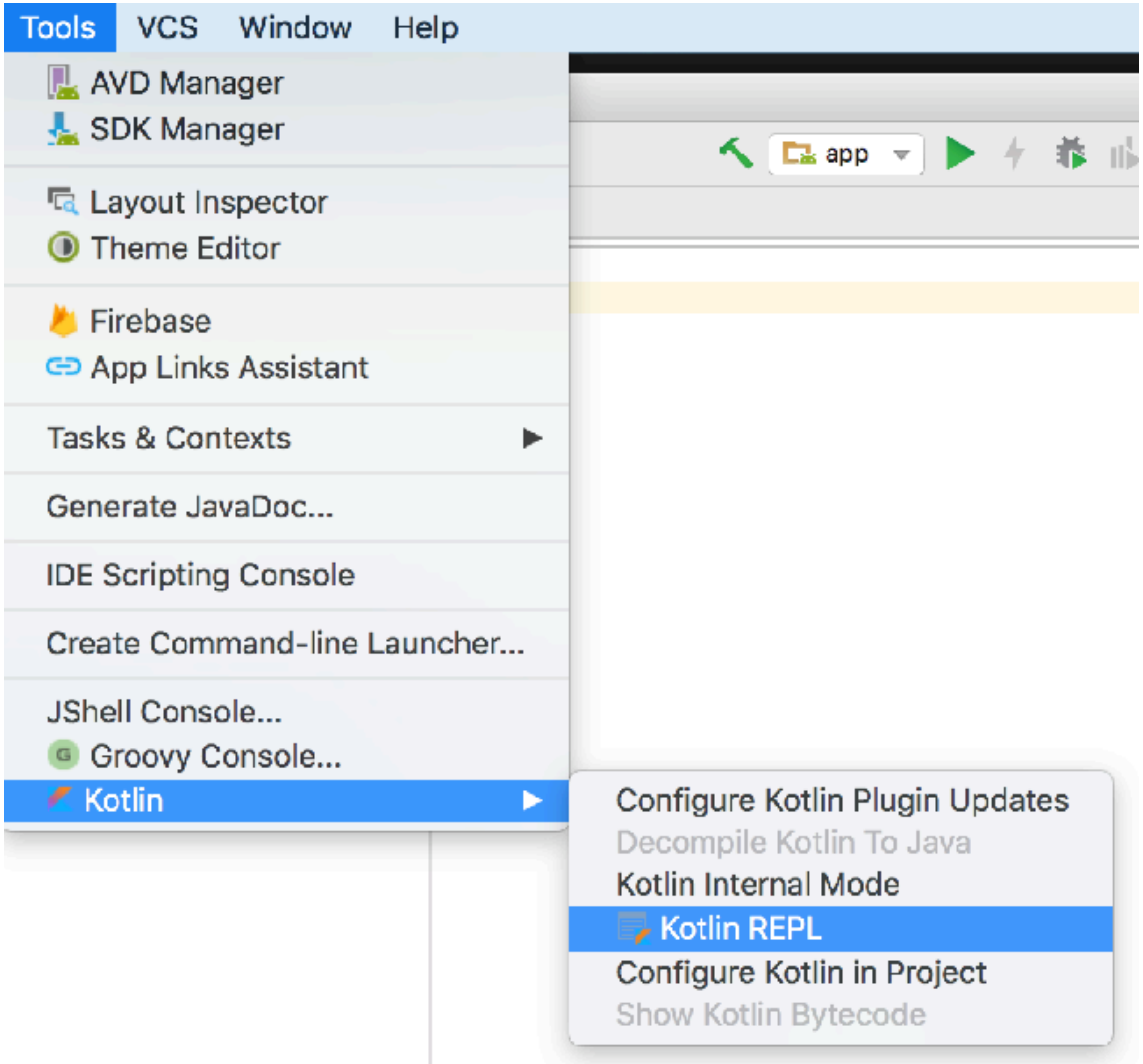
- Both statically typed
- Kotlin has type inference
    - e.g. **val x: Int = 3** vs **val x = 3**
- Java is Object-oriented, but Kotlin supports OO and functional styles

Hello World



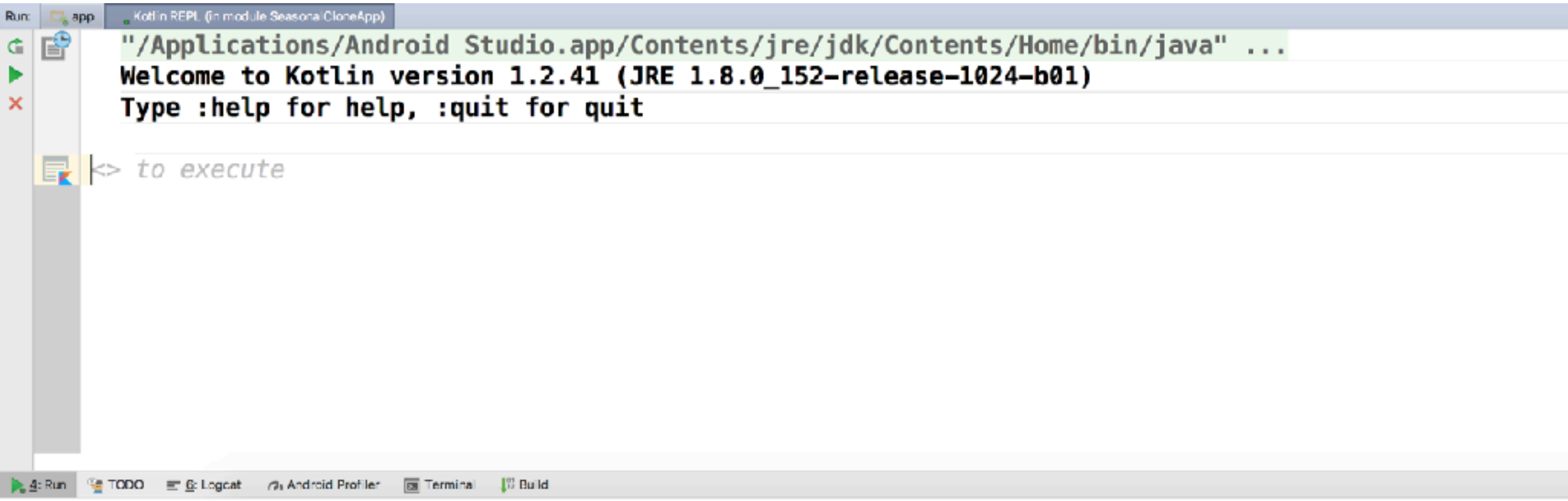
```
fun printHelloWorld () {
   println ("Hello World")
}

printHelloWorld( )
```

Kotlin REPL



@velosmobile

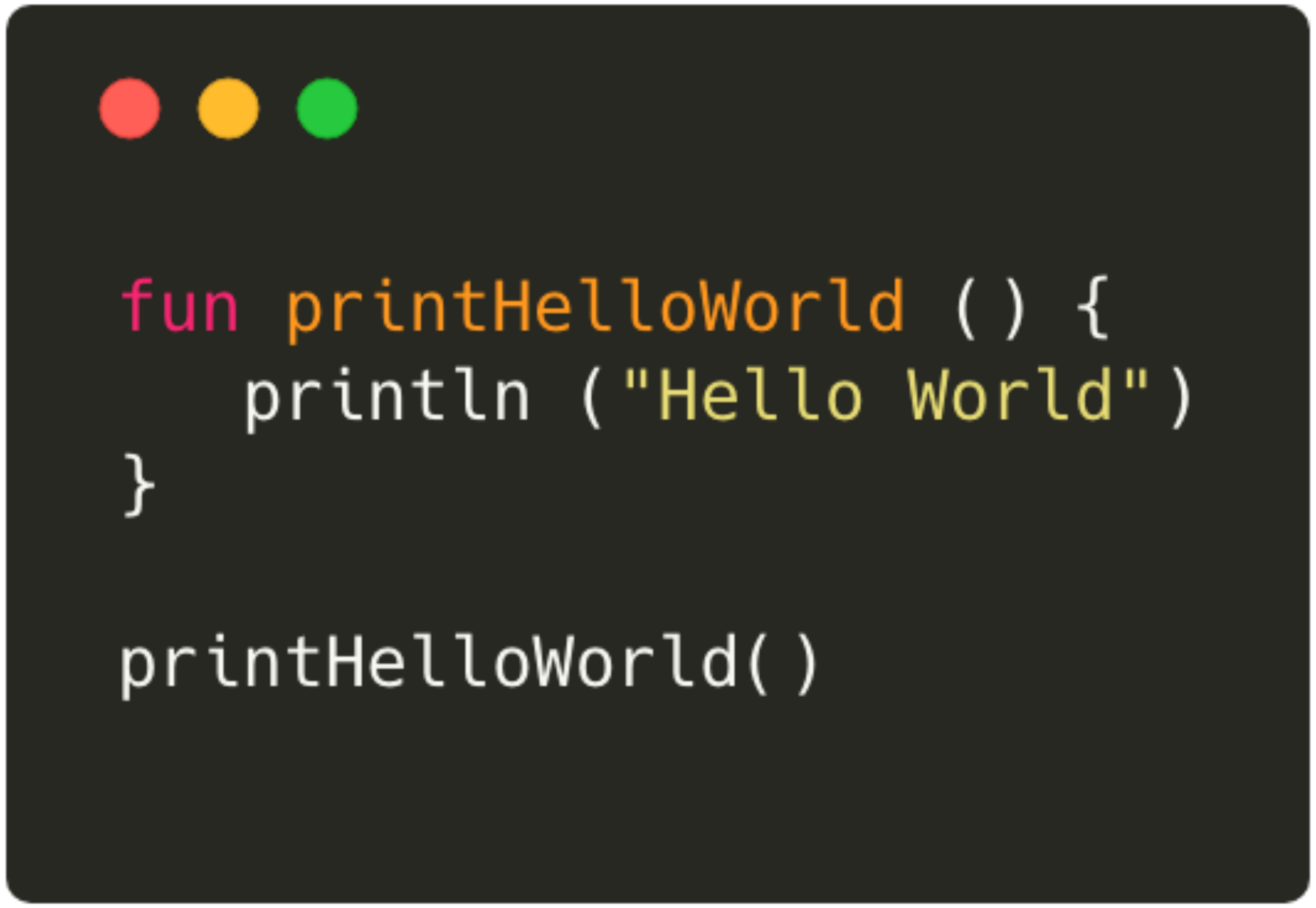
Kotlin REPL


Run: app Kotlin REPL (in module SeasonalCloneApp)
4: Run TODO 6: Logcat Android Profiler Terminal Build

```
"/Applications/Android Studio.app/Contents/jre/jdk/Contents/Home/bin/java" ...
Welcome to Kotlin version 1.2.41 (JRE 1.8.0_152-release-1024-b01)
Type :help for help, :quit for quit

<> to execute
```

@velosmobile

Hello World

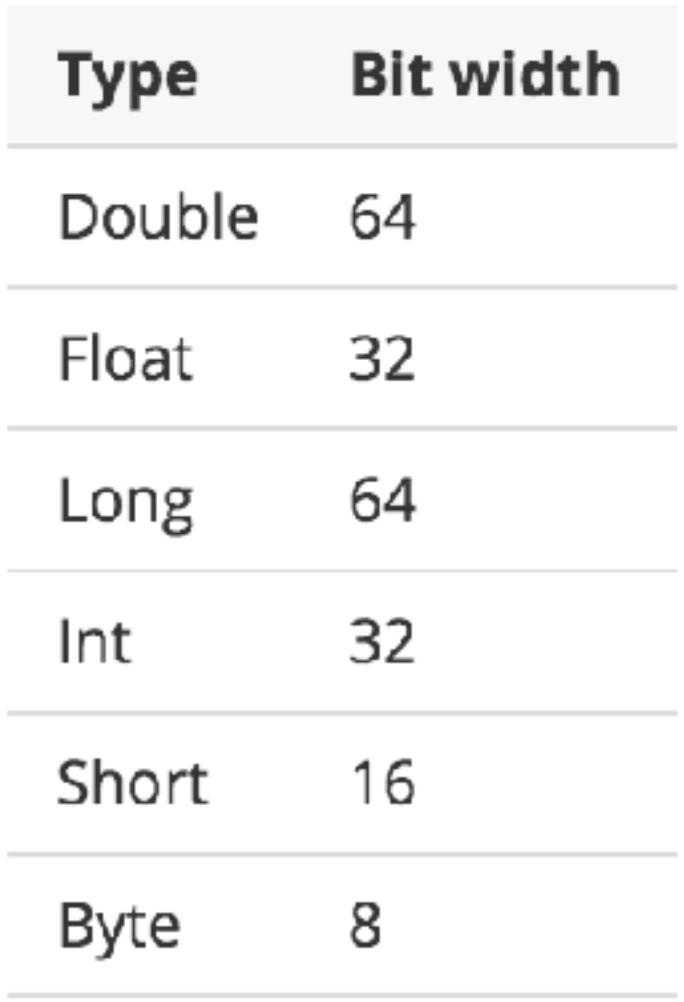


```
fun printHelloWorld () {
   println ("Hello World")
}

printHelloWorld( )
```

Objects

- Everything is treated like an object.

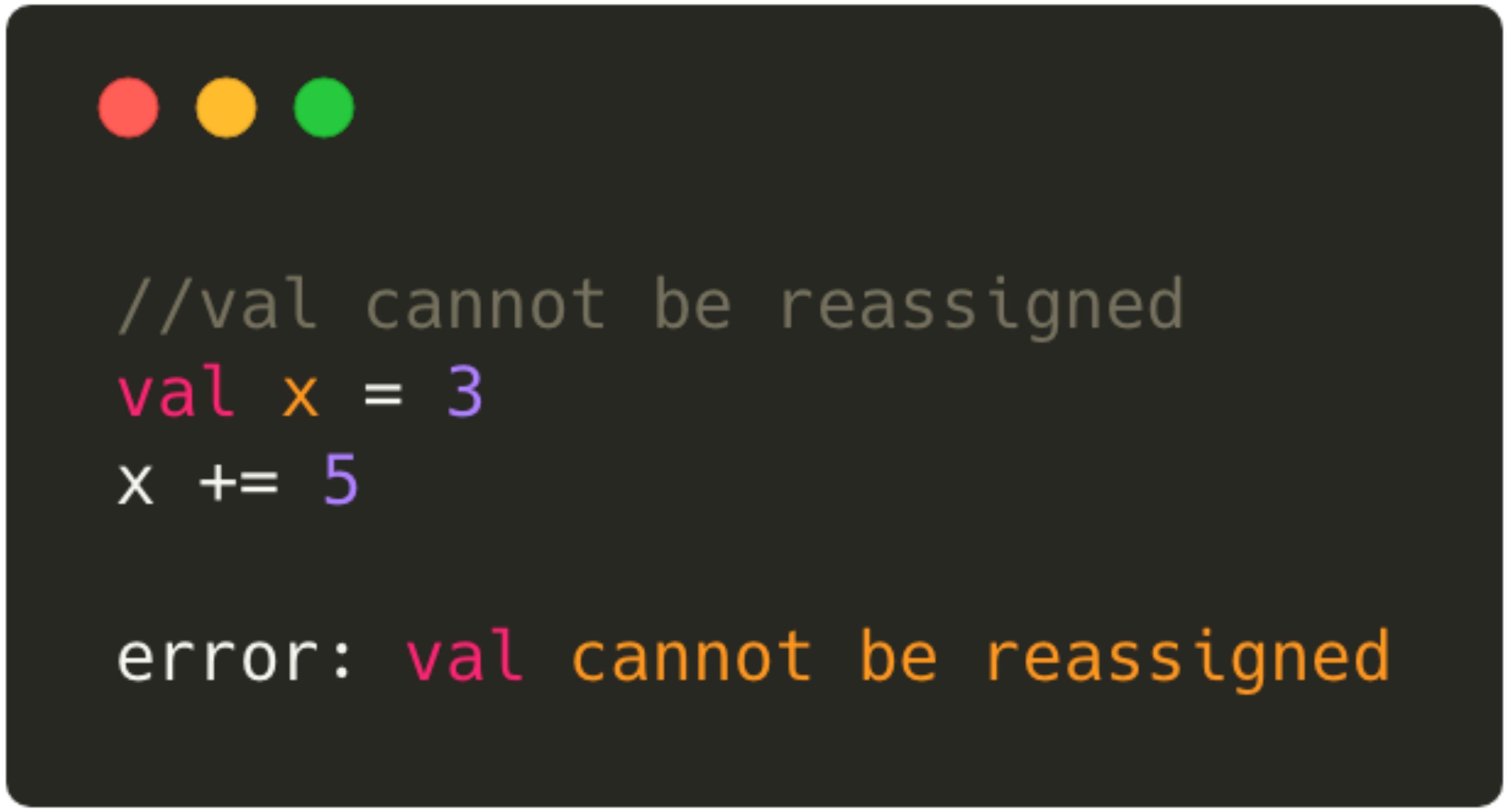
$$\begin{tikzcd}
\textbf{Any} \arrow[d] \\
\textbf{Number} \arrow[d] \\
\textbf{Int}
\end{tikzcd}$$



Type	Bit width
Double	64
Float	32
Long	64
Int	32
Short	16
Byte	8

Variables & Functions

val

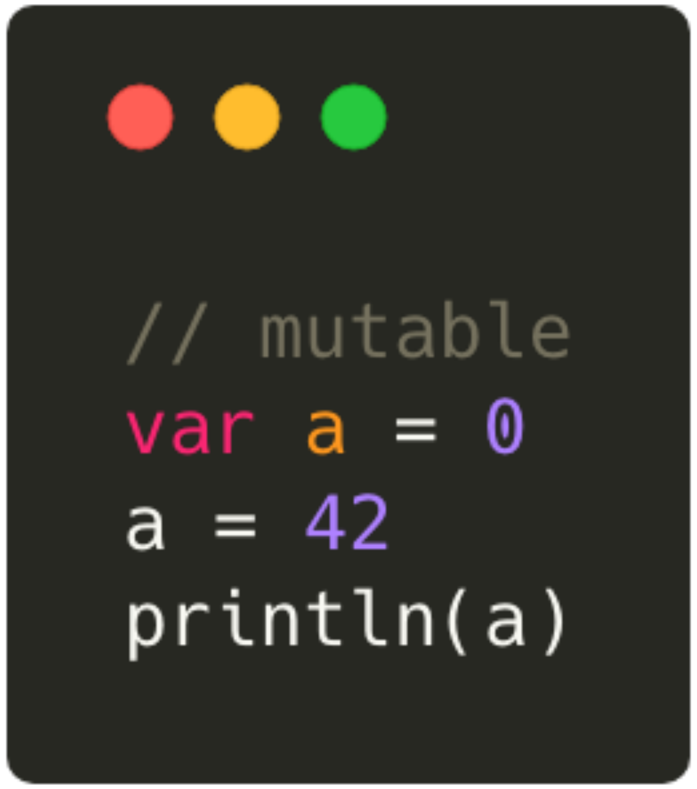


```
//val cannot be reassigned
val x = 3
x += 5

error: val cannot be reassigned
```

- **val** = final in Java
- Immutable

@velosmobile

var

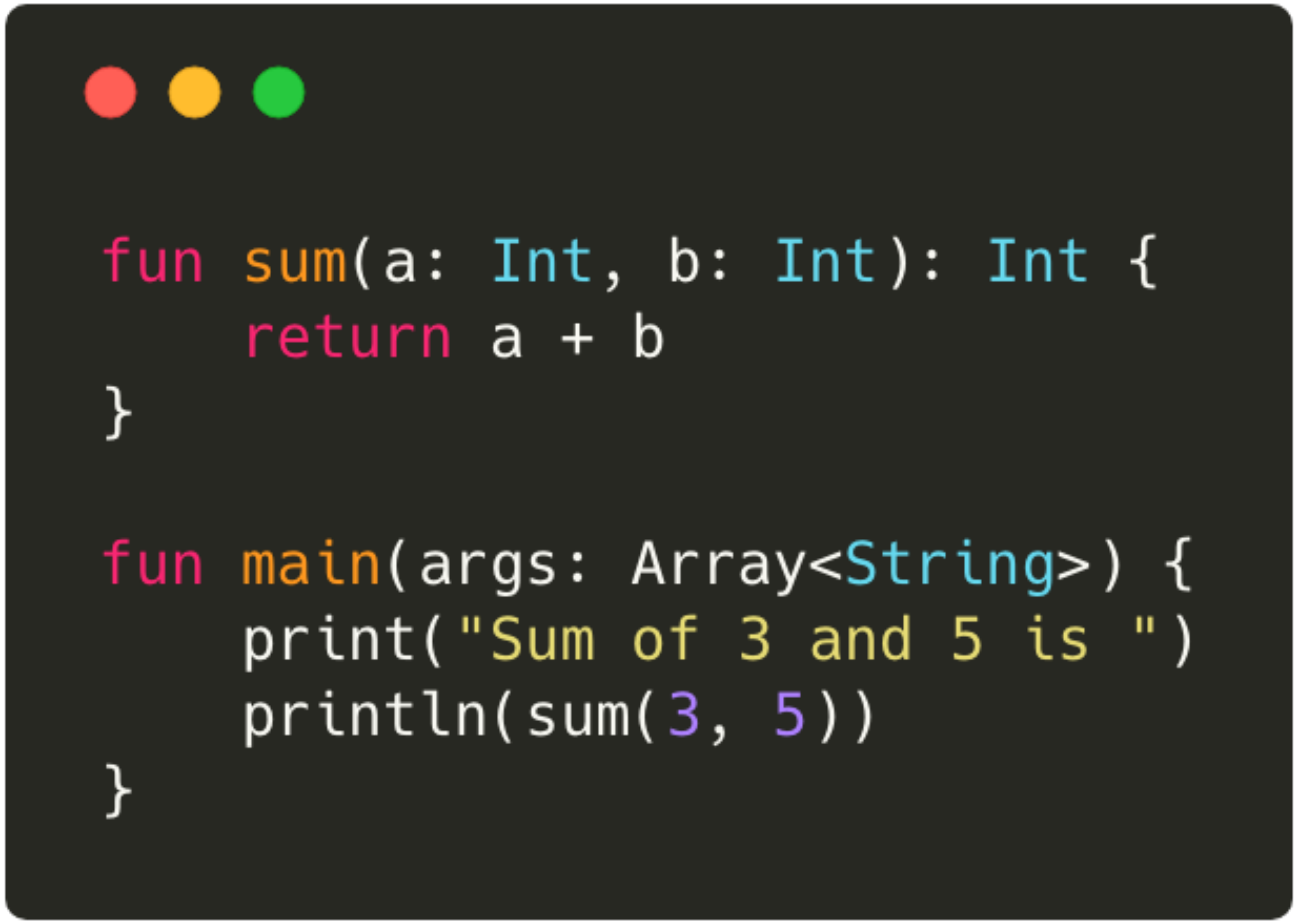


```
// mutable
var a = 0
a = 42
println(a)
```

- **var** = regular Java variable
- Mutable

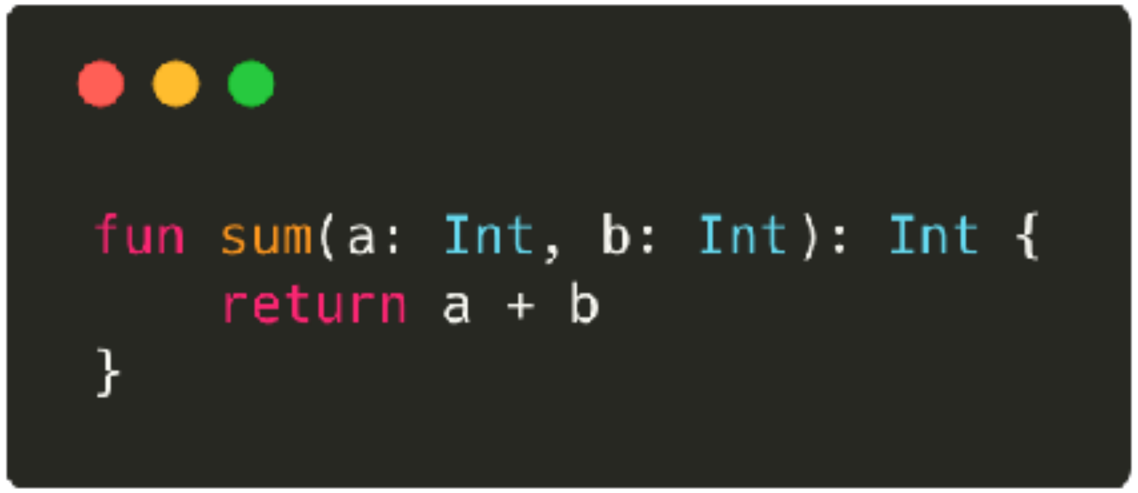
Functions

- Declared with **fun** keyword
- Return type after parameter
- Main returns 'Unit' type

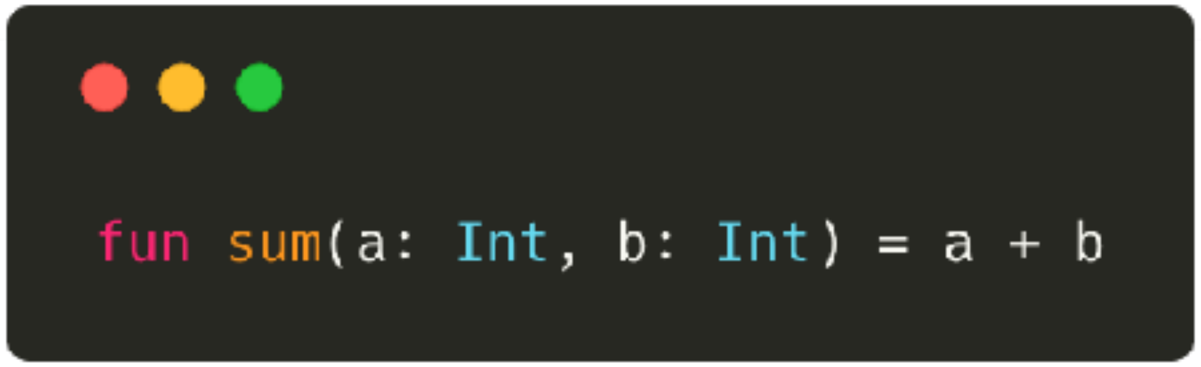


```
fun sum(a: Int, b: Int): Int {
    return a + b
}

fun main(args: Array<String>) {
    print("Sum of 3 and 5 is ")
    println(sum(3, 5))
}
```

Functions

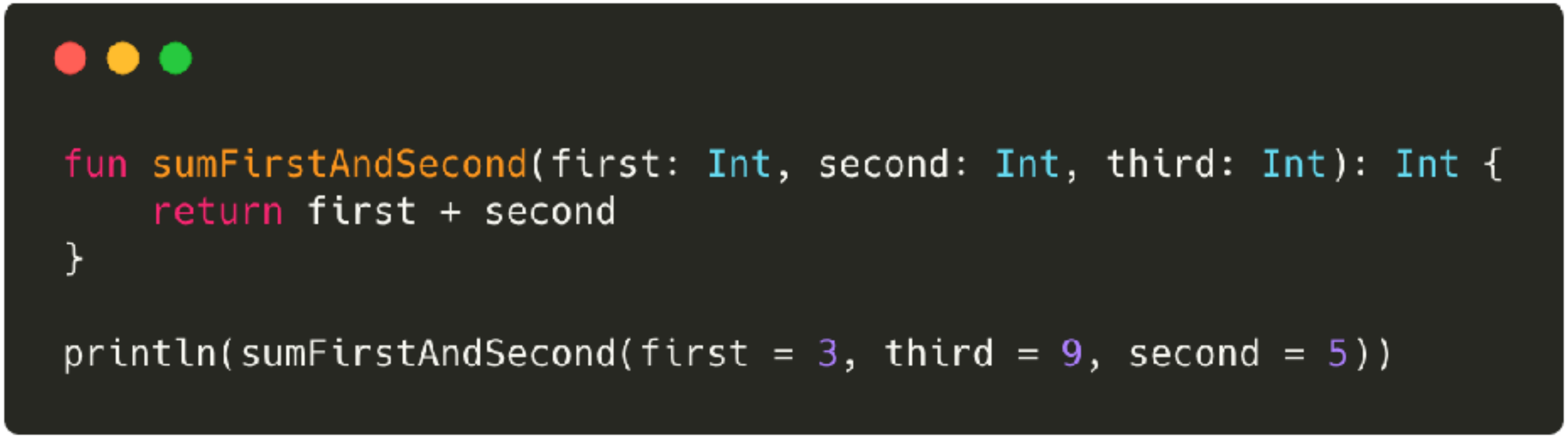

```kotlin
fun sum(a: Int, b: Int): Int {
    return a + b
}
```

=


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

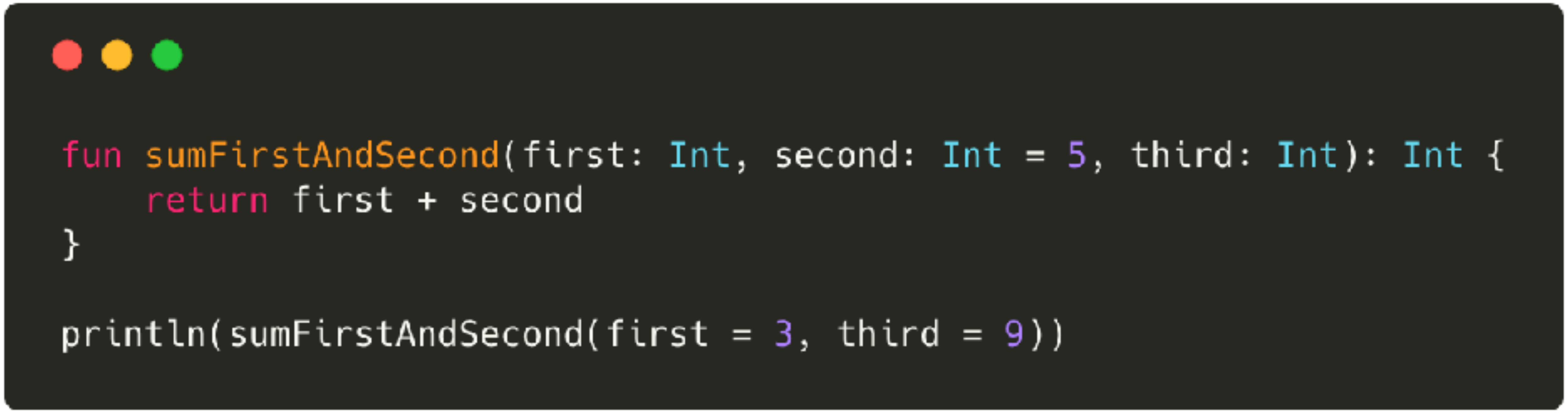
- Simplify if you're returning single expression
- Return type inferred

Named Parameters


```
fun sumFirstAndSecond(first: Int, second: Int, third: Int): Int {
    return first + second
}

println(sumFirstAndSecond(first = 3, third = 9, second = 5))
```

- Name parameters when calling method
- Call in any order

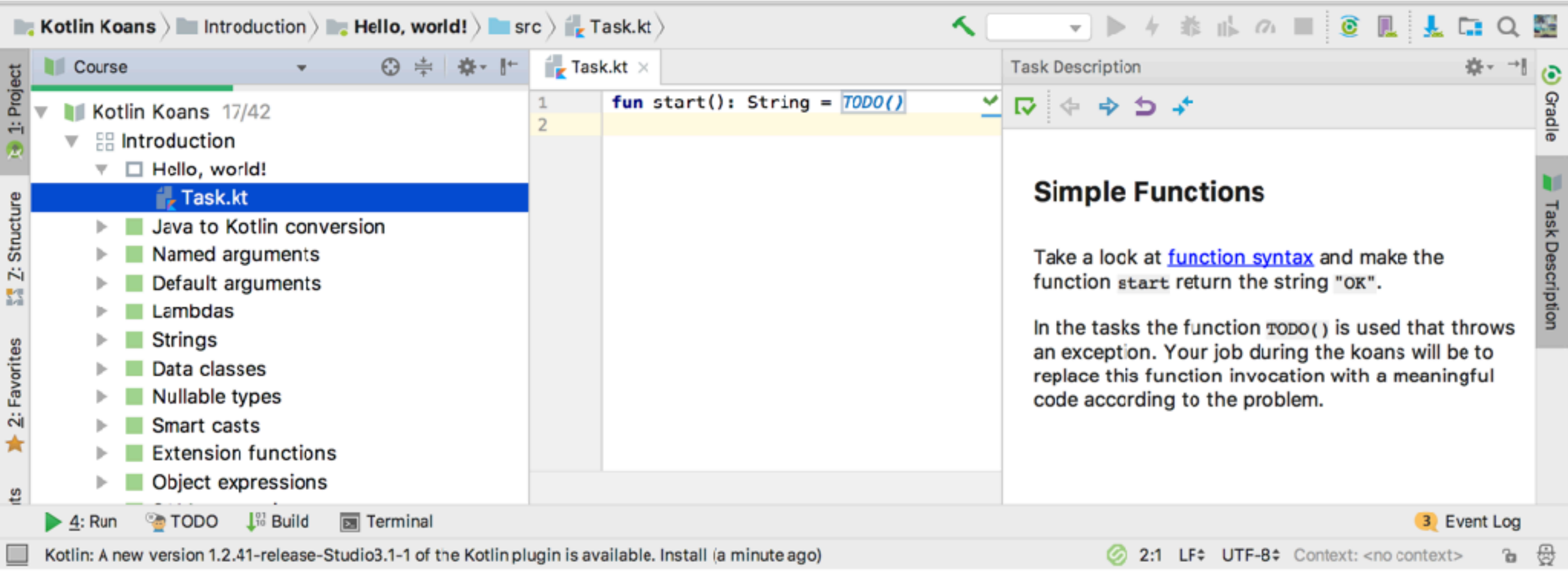
Default Parameters



```
fun sumFirstAndSecond(first: Int, second: Int = 5, third: Int): Int {
    return first + second
}

println(sumFirstAndSecond(first = 3, third = 9))
```

- Add default for a param after data type

Kotlin Koans


Kotlin Koans › Introduction › Hello, world! › src › Task.kt

Course

- Kotlin Koans 17/42
  - Introduction
    - Hello, world!
      - Task.kt
    - Java to Kotlin conversion
    - Named arguments
    - Default arguments
    - Lambdas
    - Strings
    - Data classes
    - Nullable types
    - Smart casts
    - Extension functions
    - Object expressions

Task.kt

```
fun start(): String = TODO()
```

Task Description

Simple Functions

Take a look at function syntax and make the function `start` return the string `"OK"`.

In the tasks the function `TODO()` is used that throws an exception. Your job during the koans will be to replace this function invocation with a meaningful code according to the problem.

Kotlin: A new version 1.2.41-release-Studio3.1-1 of the Kotlin plugin is available. Install (a minute ago)

2:1 LF‡ UTF-8‡ Context: <no context>

https://kotlinlang.org/docs/tutorials/edu-tools-learner.html

Koans

20min

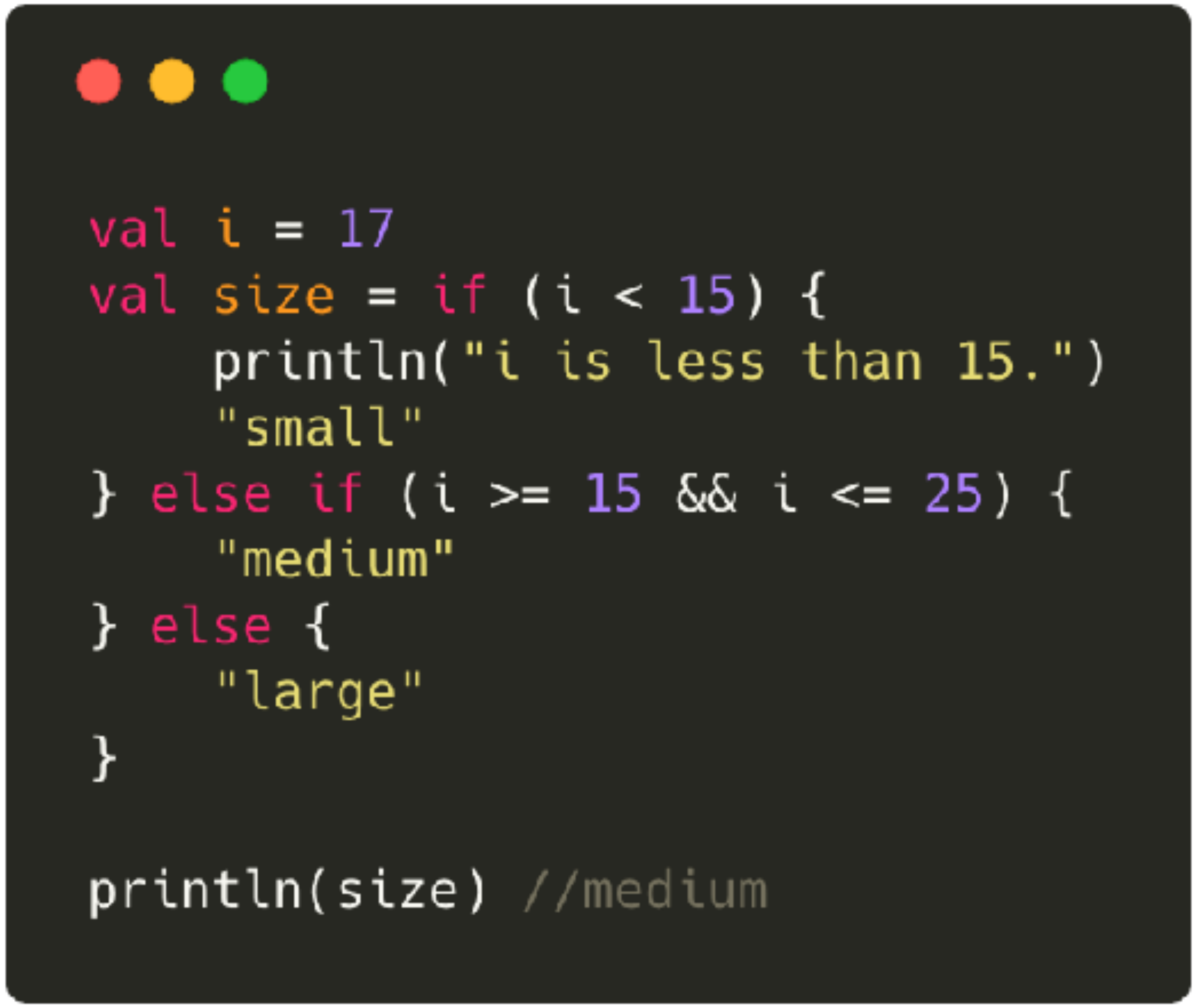
Introduction:

- Hello, world!
- Java to Kotlin conversion
- Default arguments
- (Bonus) Named arguments

Control Structures

if

- Expressions, not statements
- Assign value on last line
- Default return type is “Unit” (like void in Java)

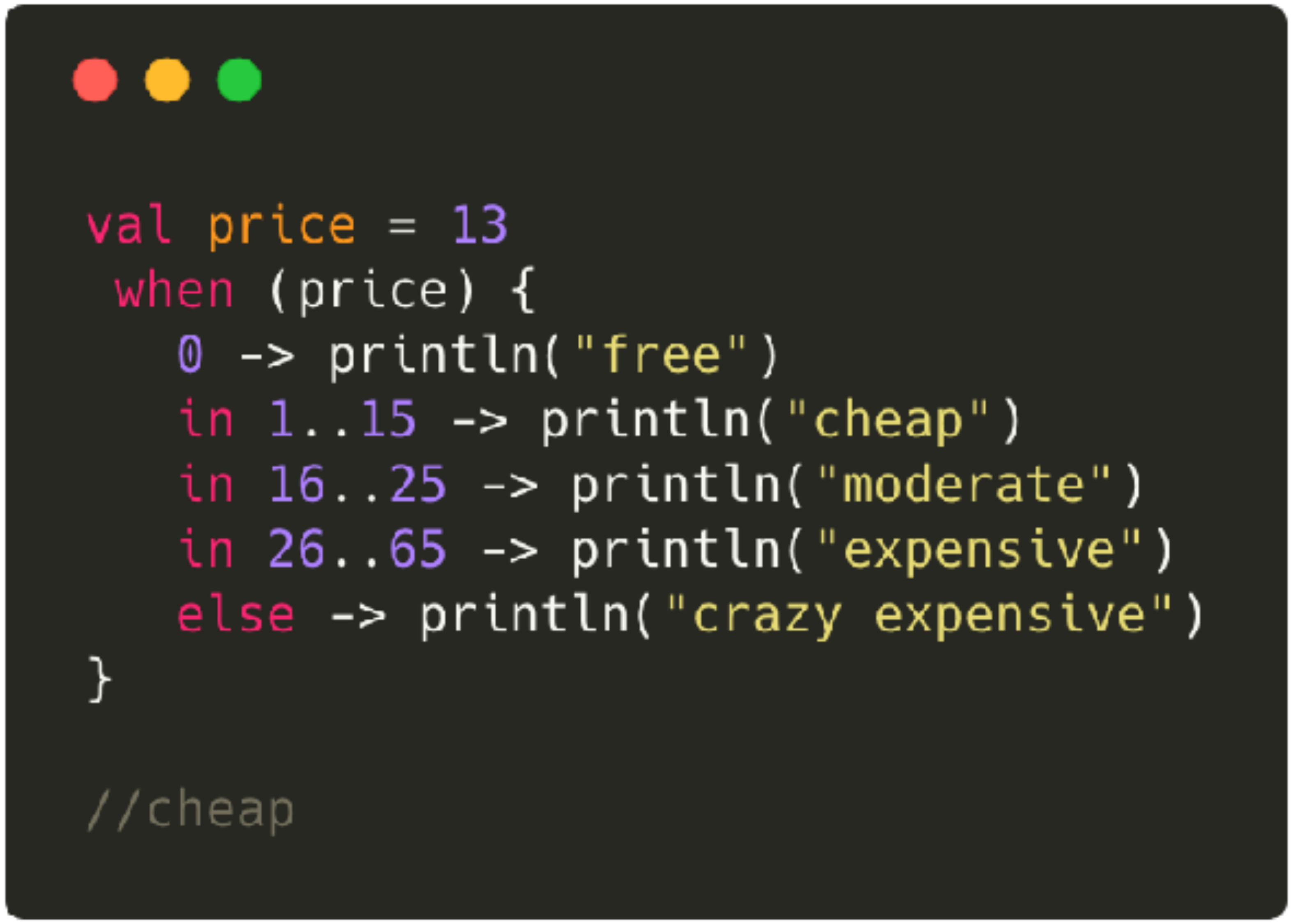


```
val i = 17
val size = if (i < 15) {
    println("i is less than 15.")
    "small"
} else if (i >= 15 && i <= 25) {
    "medium"
} else {
    "large"
}

println(size) //medium
```

when

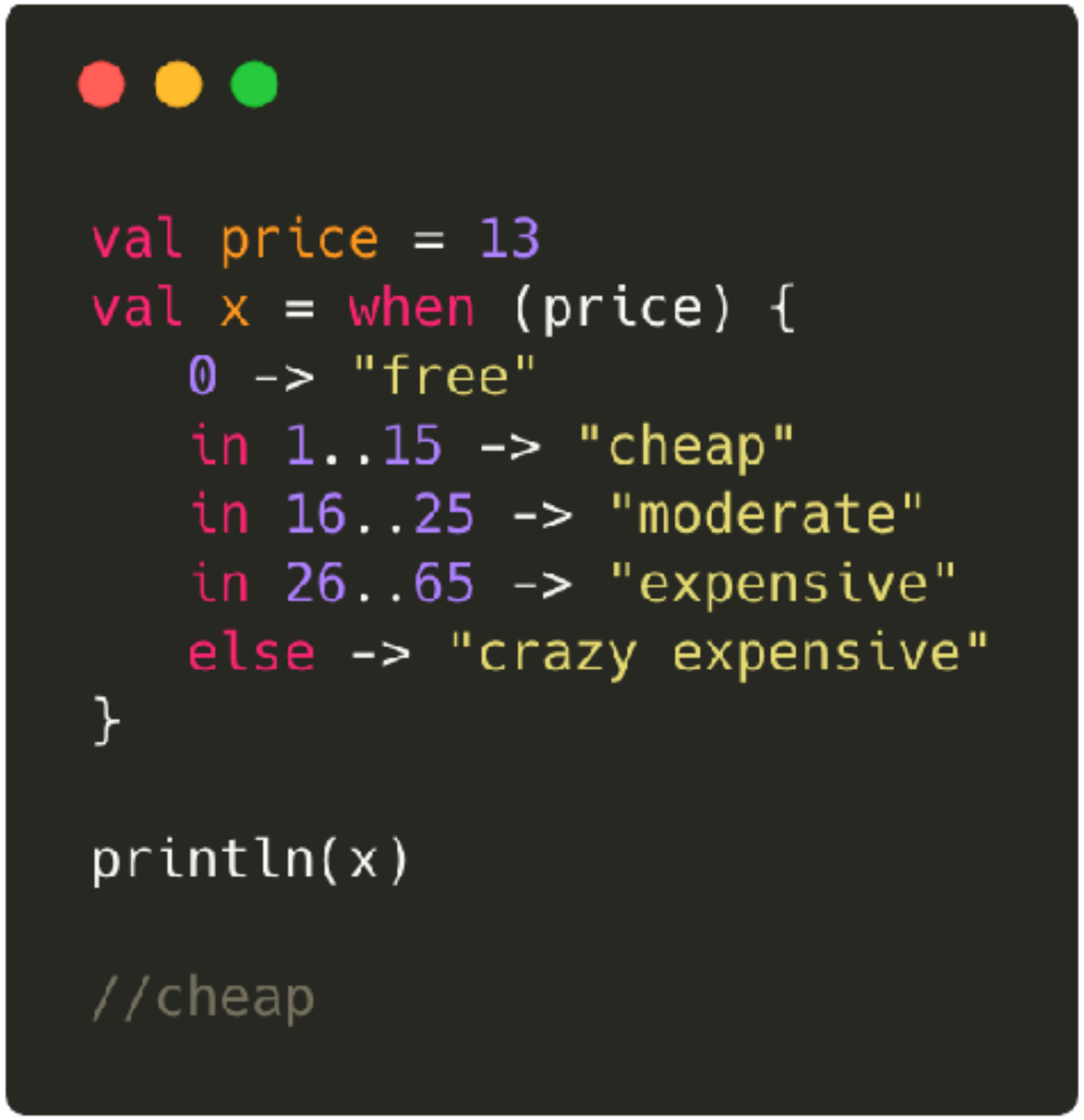
- Similar to switch-case

- Checks if left hand side evaluates to true



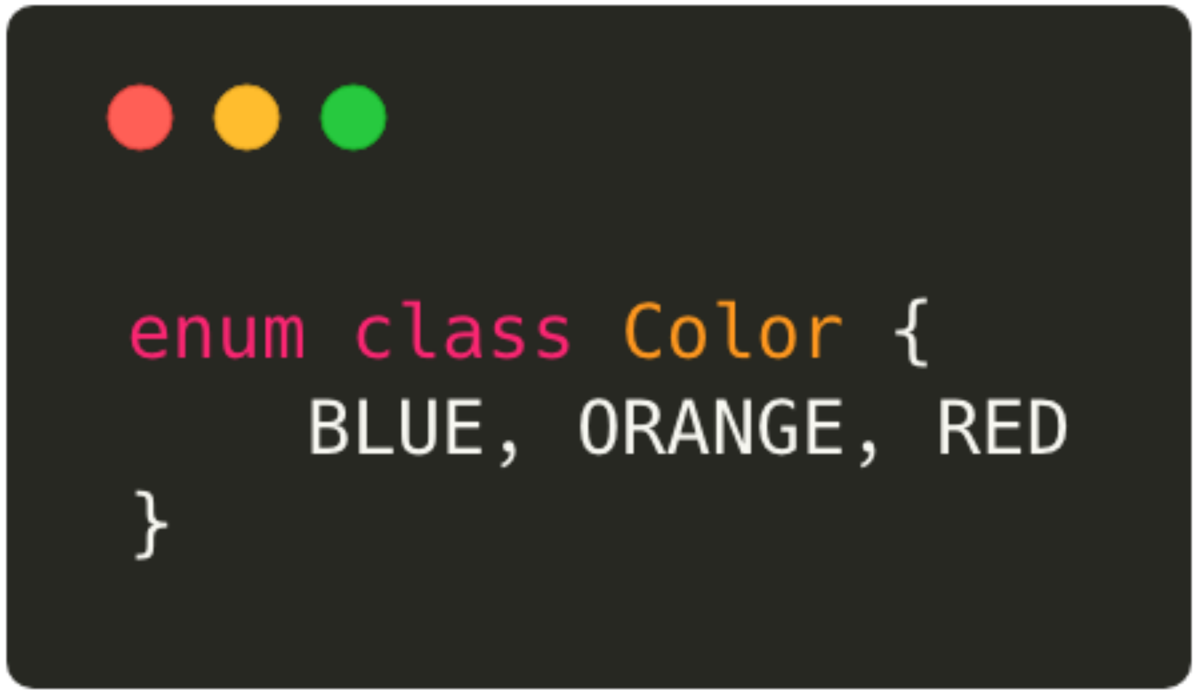
```
val price = 13
 when (price) {
   0 -> println("free")
   in 1..15 -> println("cheap")
   in 16..25 -> println("moderate")
   in 26..65 -> println("expensive")
   else -> println("crazy expensive")
}

//cheap
```

when



```
val price = 13
val x = when (price) {
    0 -> "free"
    in 1..15 -> "cheap"
    in 16..25 -> "moderate"
    in 26..65 -> "expensive"
    else -> "crazy expensive"
}

println(x)

//cheap
```

enums

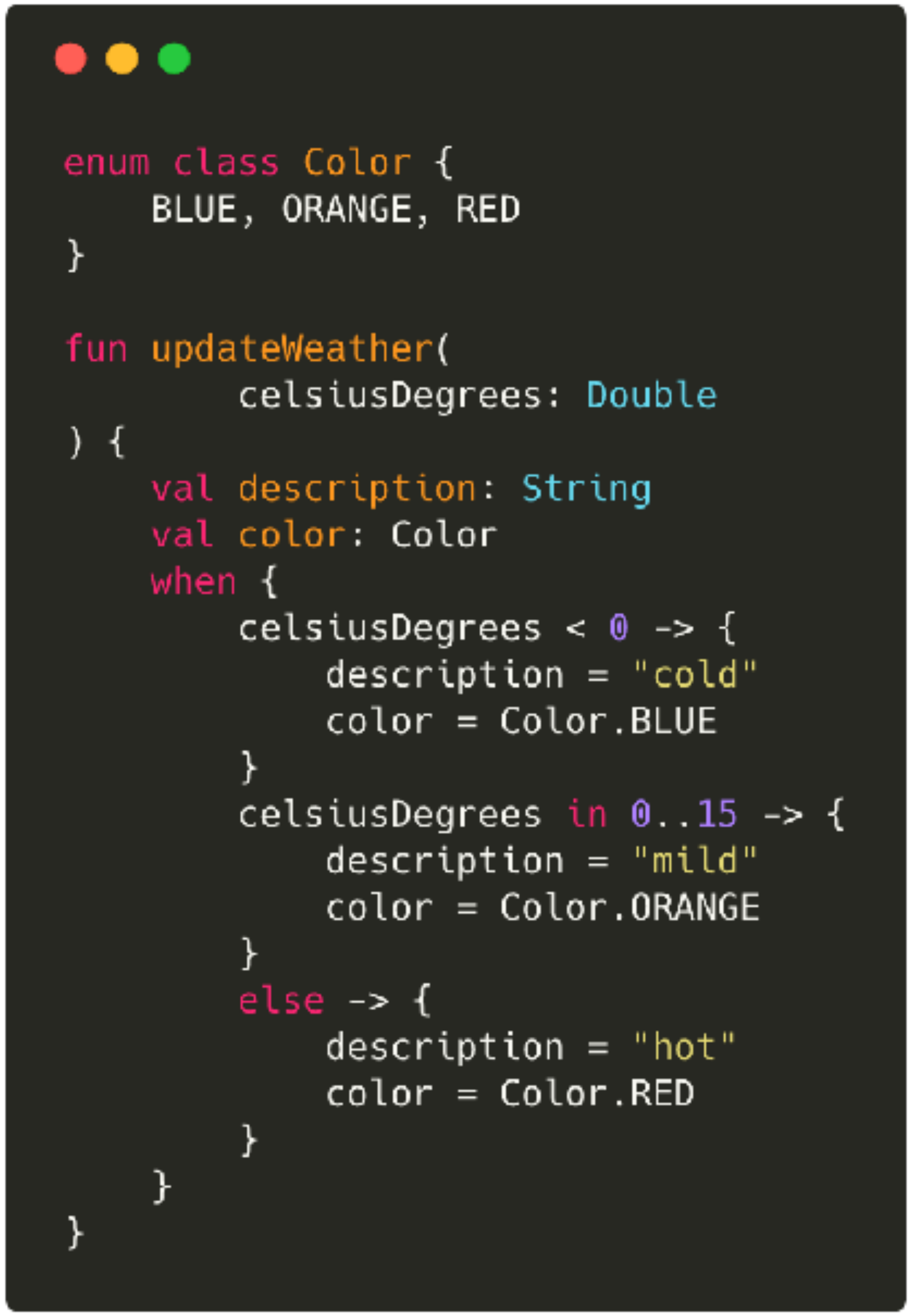


```kotlin
enum class Color {
    BLUE, ORANGE, RED
}
```

- Keyword **enum** for enum classes

enums and when

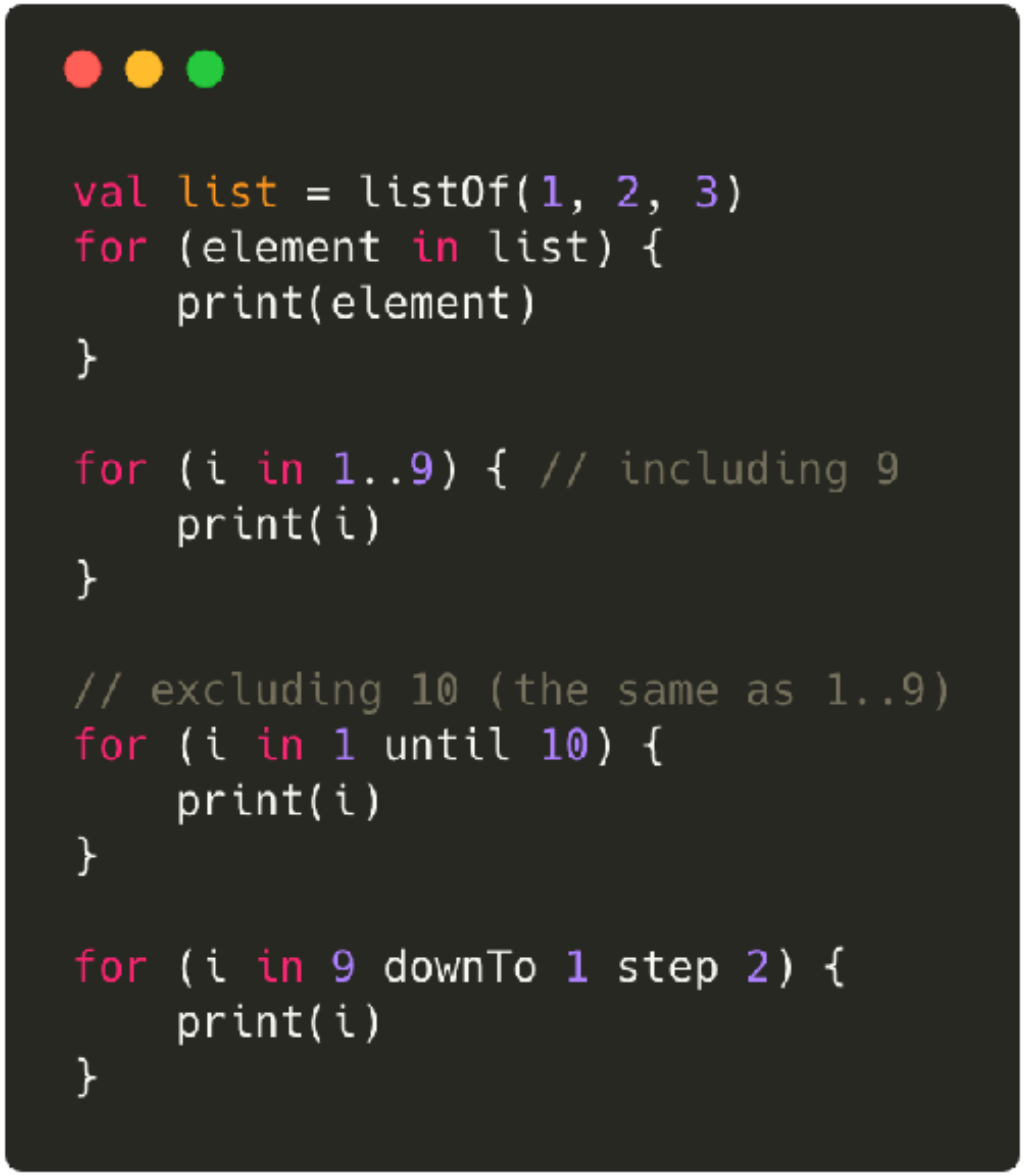
- Use enums with when expressions
- Can use when without a condition



```
enum class Color {
    BLUE, ORANGE, RED
}

fun updateWeather(
        celsiusDegrees: Double
) {
    val description: String
    val color: Color
    when {
        celsiusDegrees < 0 -> {
            description = "cold"
            color = Color.BLUE
        }
        celsiusDegrees in 0..15 -> {
            description = "mild"
            color = Color.ORANGE
        }
        else -> {
            description = "hot"
            color = Color.RED
        }
    }
}
```

@velosmobile

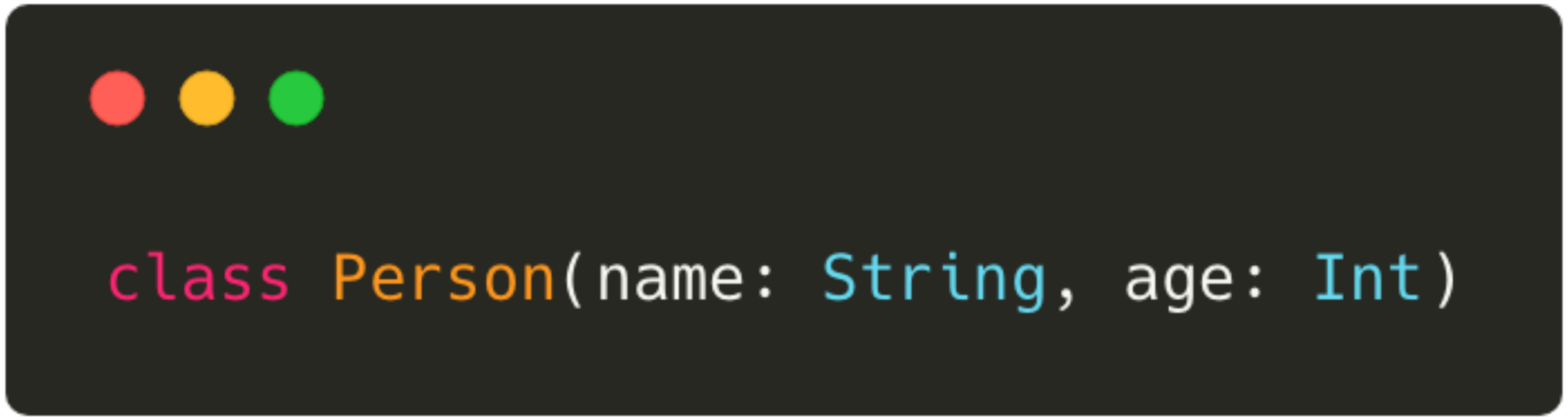
Loops



```
val list = listOf(1, 2, 3)
for (element in list) {
    print(element)
}

for (i in 1..9) { // including 9
    print(i)
}

// excluding 10 (the same as 1..9)
for (i in 1 until 10) {
    print(i)
}

for (i in 9 downTo 1 step 2) {
    print(i)
}
```

Classes

Classes



```
class Person(name: String, age: Int)
```

- Don’t need **new** keyword for new instance:

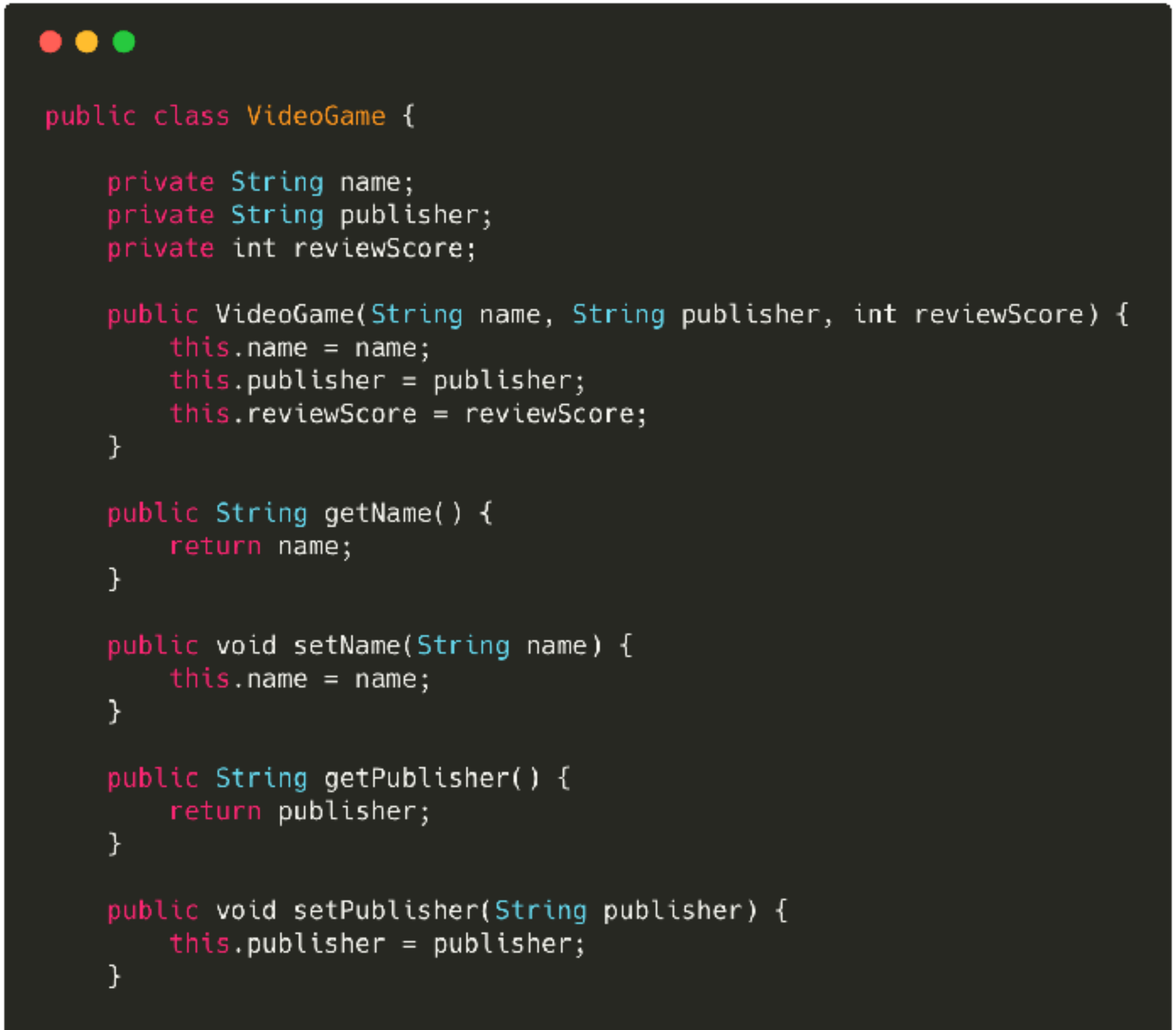
  val person = Person(“John”, 55)

- Implicit calls to getters/setters

Data Classes

- Replace POJOs in Java
- Keyword **data** in front of class
- Lets you access:
  - hashCode()
  - equals()
  - toString()
  - copy()

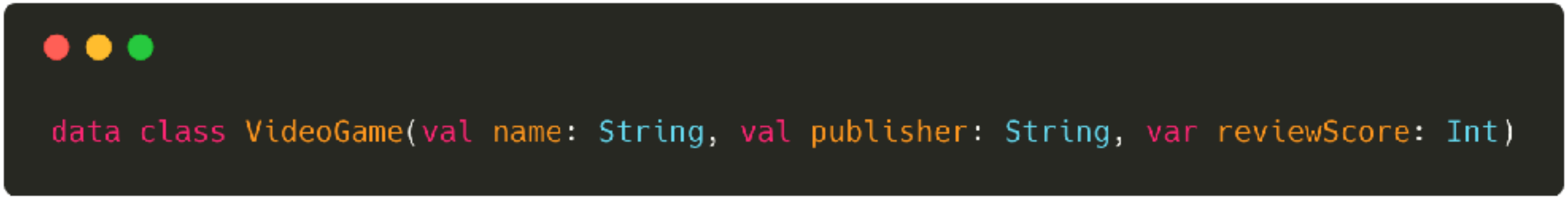
Data Classes Example



```java
public class VideoGame {

    private String name;
    private String publisher;
    private int reviewScore;

    public VideoGame(String name, String publisher, int reviewScore) {
        this.name = name;
        this.publisher = publisher;
        this.reviewScore = reviewScore;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getPublisher() {
        return publisher;
    }

    public void setPublisher(String publisher) {
        this.publisher = publisher;
    }
```

@velosmobile

Data Classes Example



```kotlin
data class VideoGame(val name: String, val publisher: String, var reviewScore: Int)
```

Source: Data Classes in Kotlin

val videogame = VideoGame("a name", "a publisher", 4)

- Use implicit getters/setters

videogame.name = "some game"

videogame.name

Koans

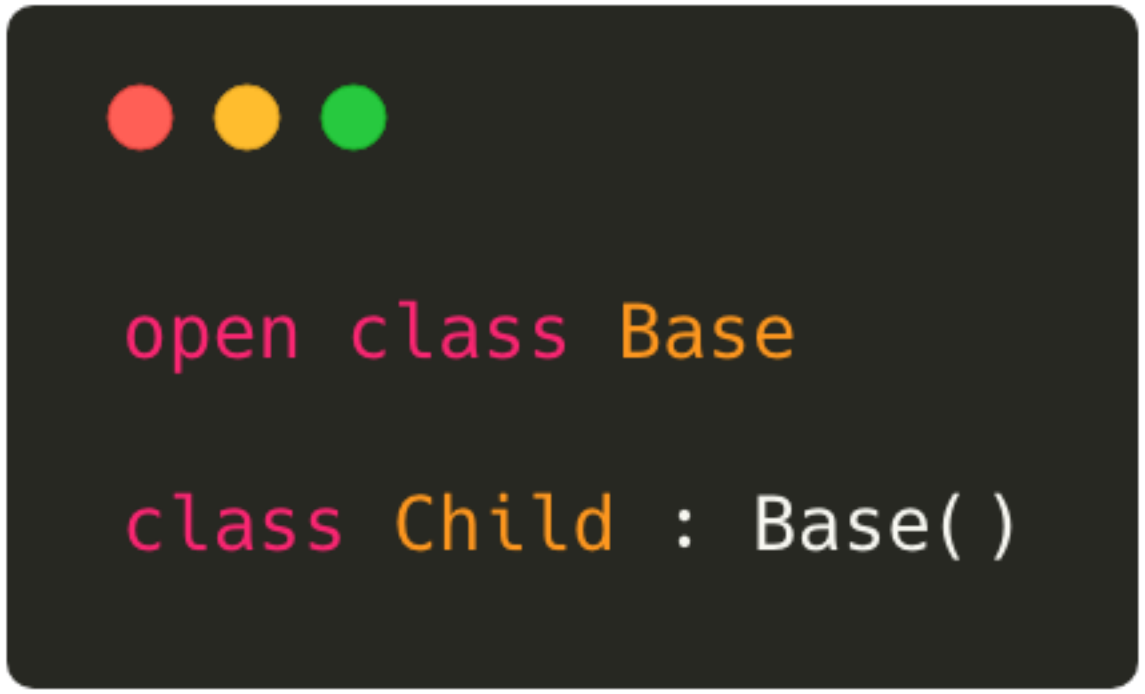
10min

Introduction:

- Data Classes

Hierarchies

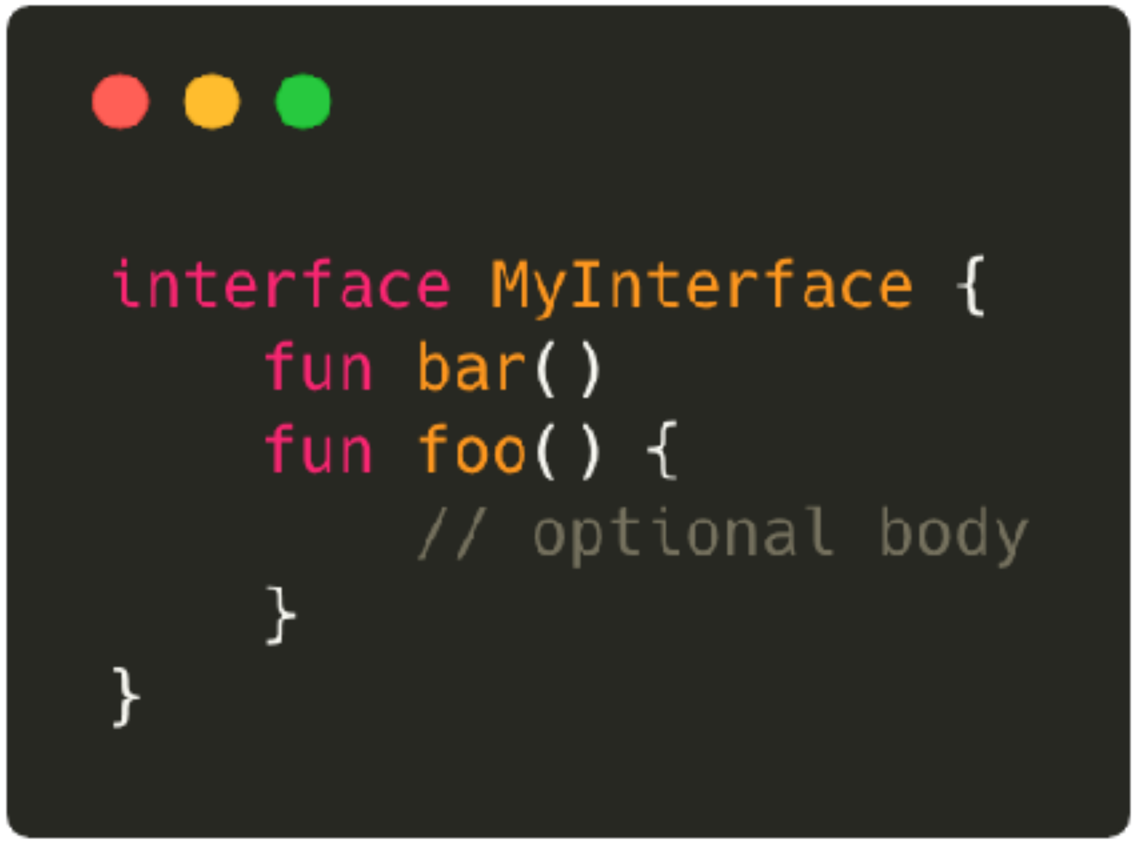
Open Classes



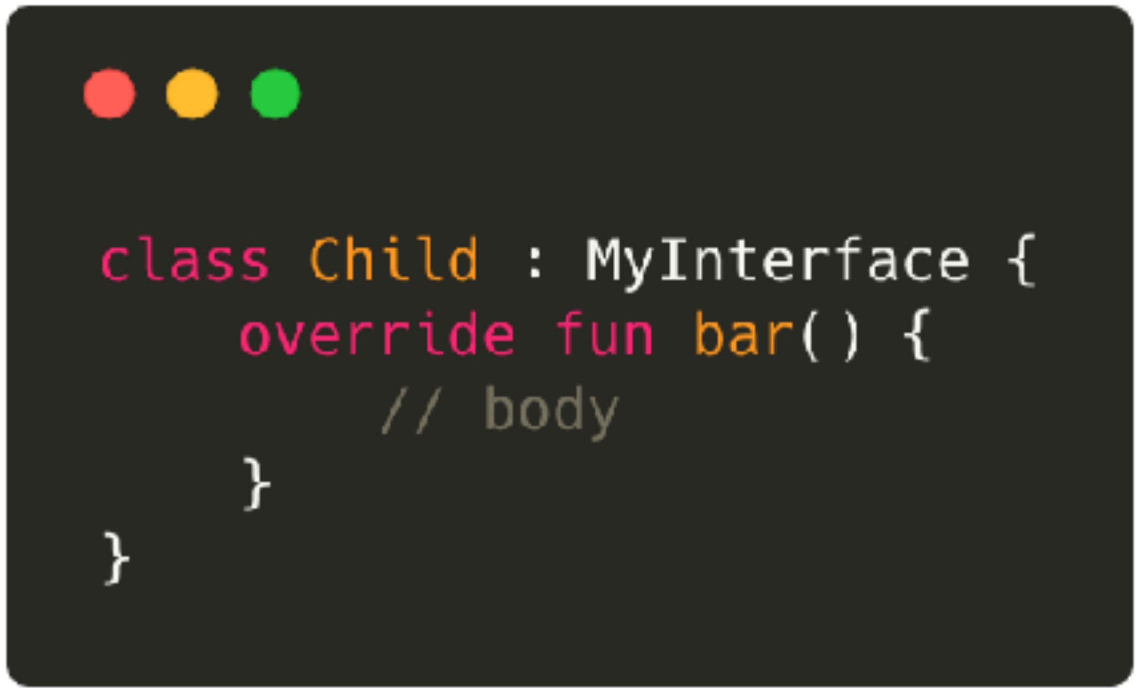
```
open class Base

class Child : Base()
```

- Classes in Kotlin by default are closed
- Think, “Should this class be inheritable?”

Interfaces



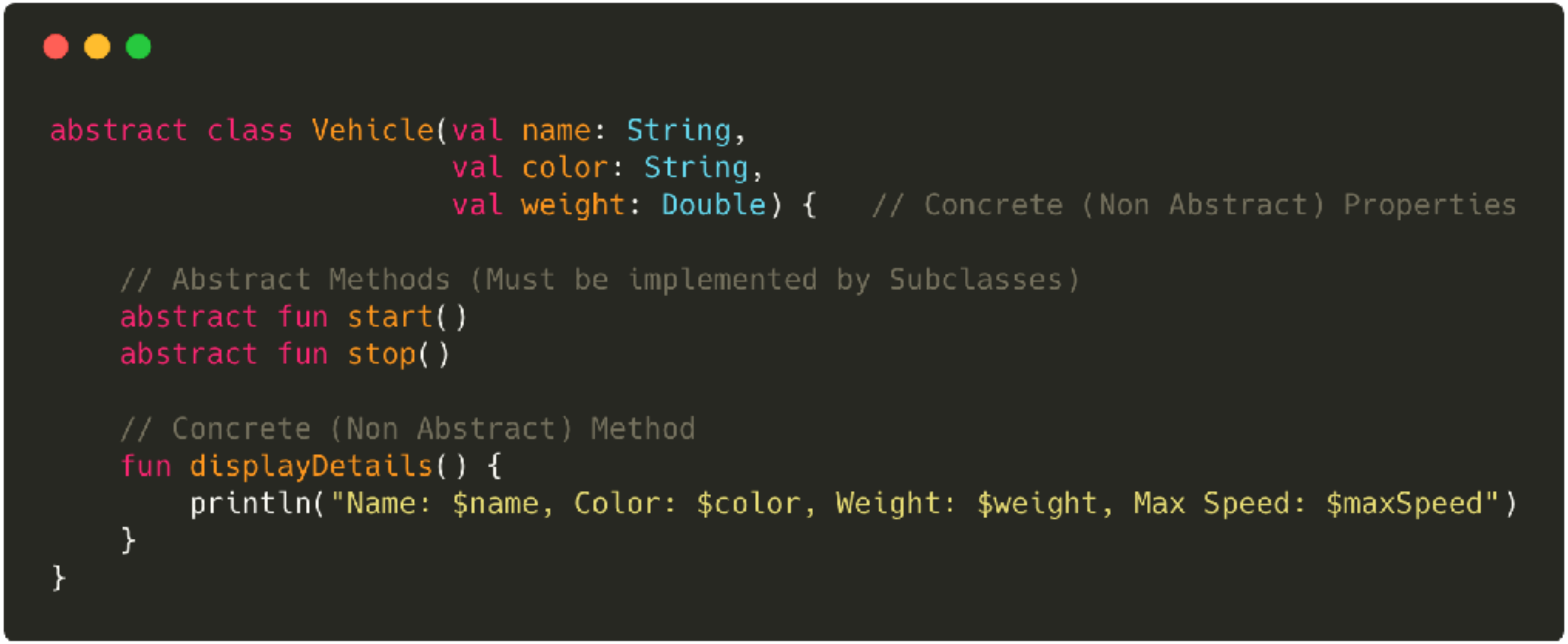
```
interface MyInterface {
     fun bar()
     fun foo() {
          // optional body
     }
}
```



```
class Child : MyInterface {
     override fun bar() {
          // body
     }
}
```

- Use : for implements
- Must override bar()
- foo() is optional

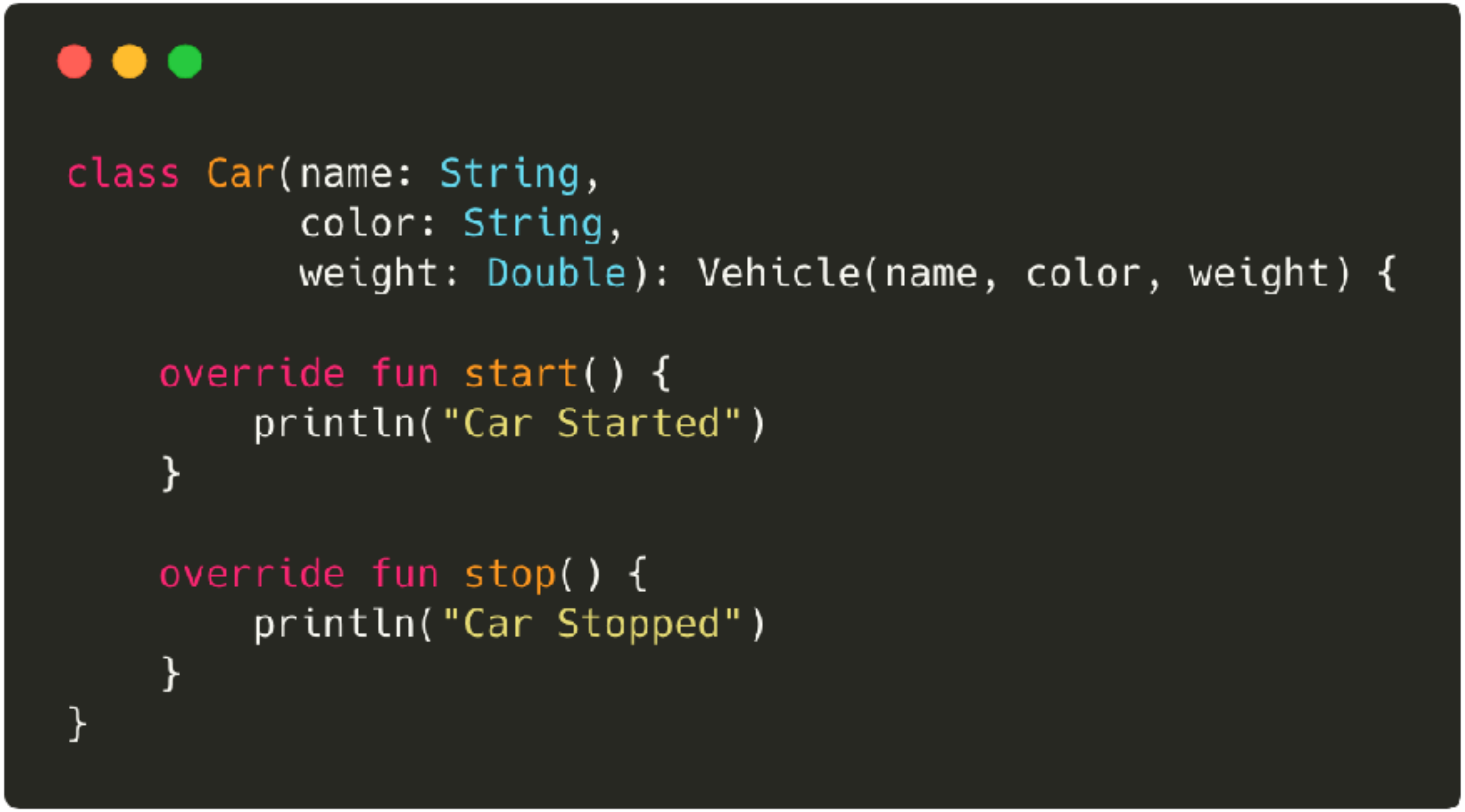
Abstract Classes



```kotlin
abstract class Vehicle(val name: String,
                       val color: String,
                       val weight: Double) {   // Concrete (Non Abstract) Properties

    // Abstract Methods (Must be implemented by Subclasses)
    abstract fun start()
    abstract fun stop()

    // Concrete (Non Abstract) Method
    fun displayDetails() {
        println("Name: $name, Color: $color, Weight: $weight, Max Speed: $maxSpeed")
    }
}
```

Source: Kotlin Abstract Classes

Abstract Classes

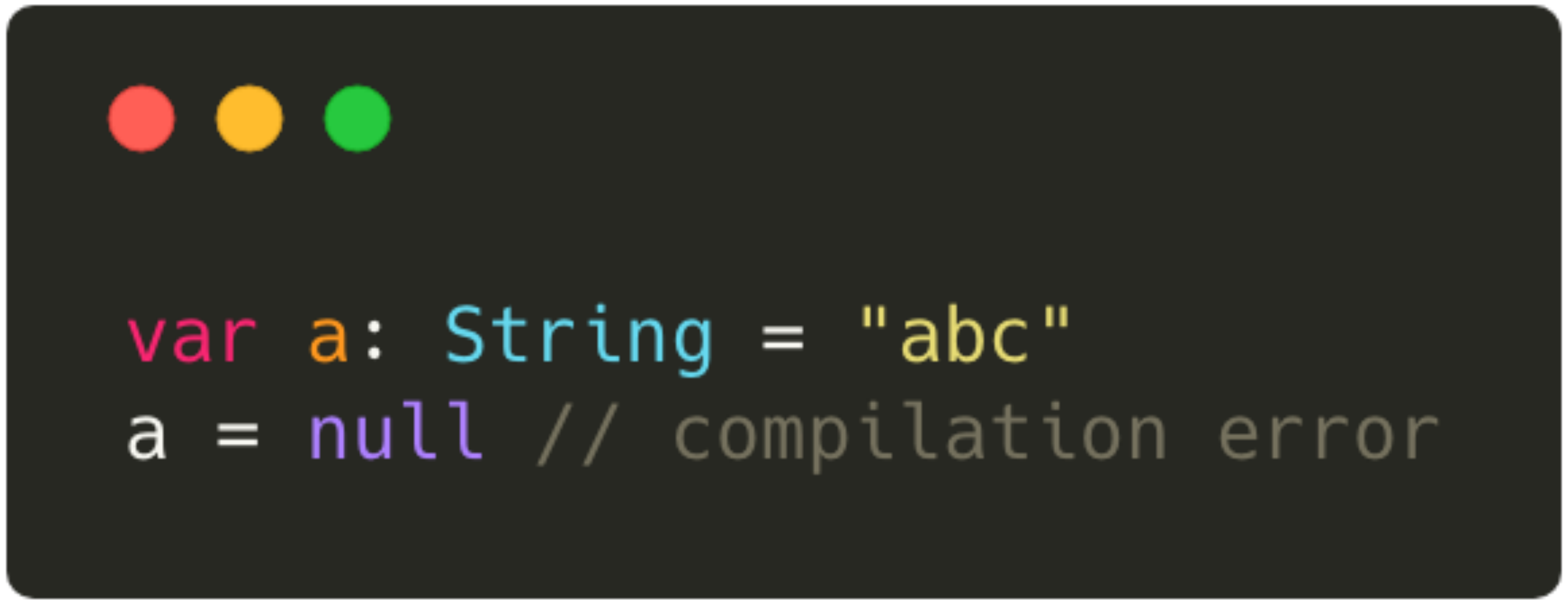


```kotlin
class Car(name: String,
          color: String,
          weight: Double): Vehicle(name, color, weight) {

    override fun start() {
        println("Car Started")
    }

    override fun stop() {
        println("Car Stopped")
    }
}
```

- Use : to extend

Nullability

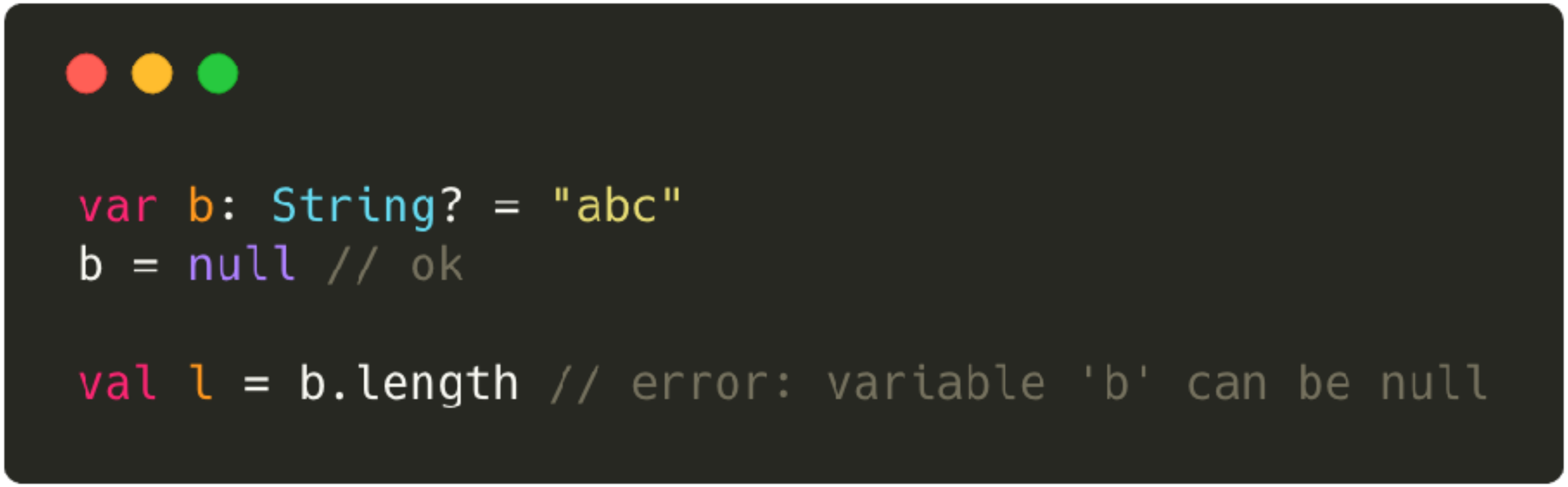
Null Safety



```
var a: String = "abc"
a = null // compilation error
```

- Error caught at compile time
- **a** can't be null here

Nullability

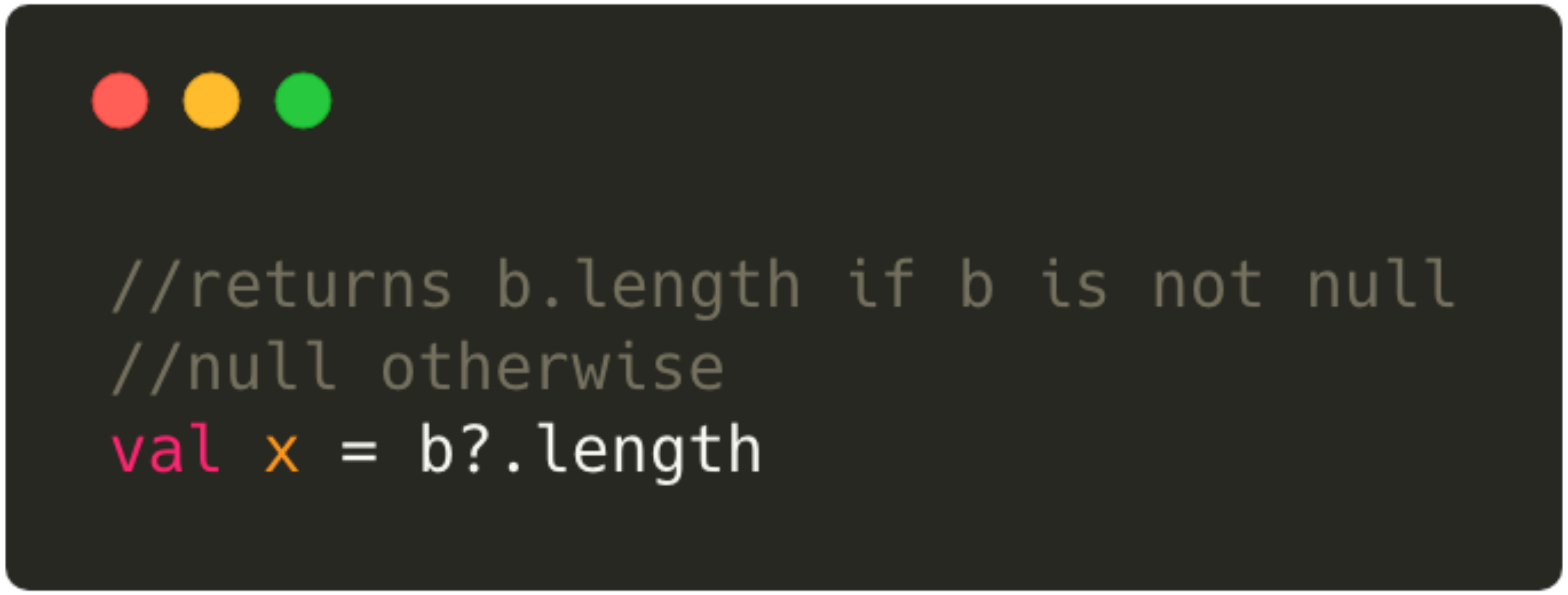


```
var b: String? = "abc"
b = null // ok

val l = b.length // error: variable 'b' can be null
```

- Use ? To declare nullable
- b.length doesn't compile

Safe Calls

?.

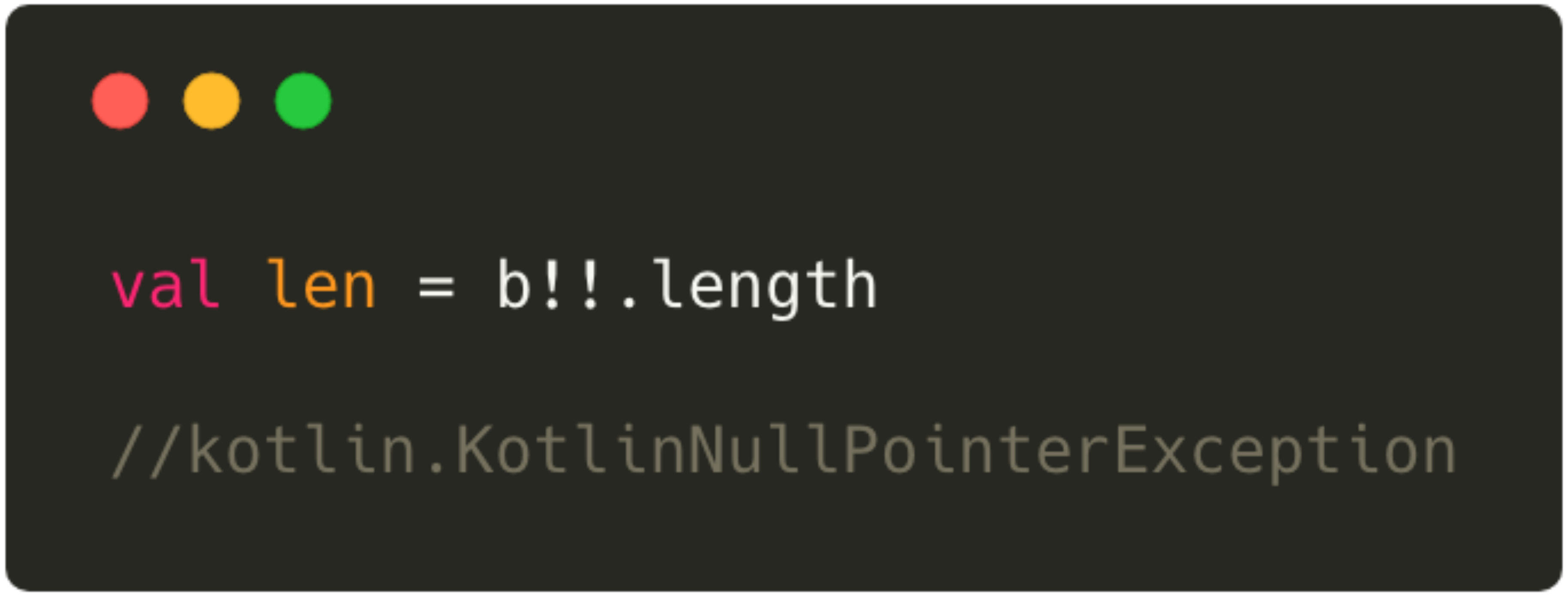


```
//returns b.length if b is not null
//null otherwise
val x = b?.length
```

- Use ? to check if null
- Returns b.length if b is not null; null otherwise
- Type of x is **Int?**

Non-null Asserted

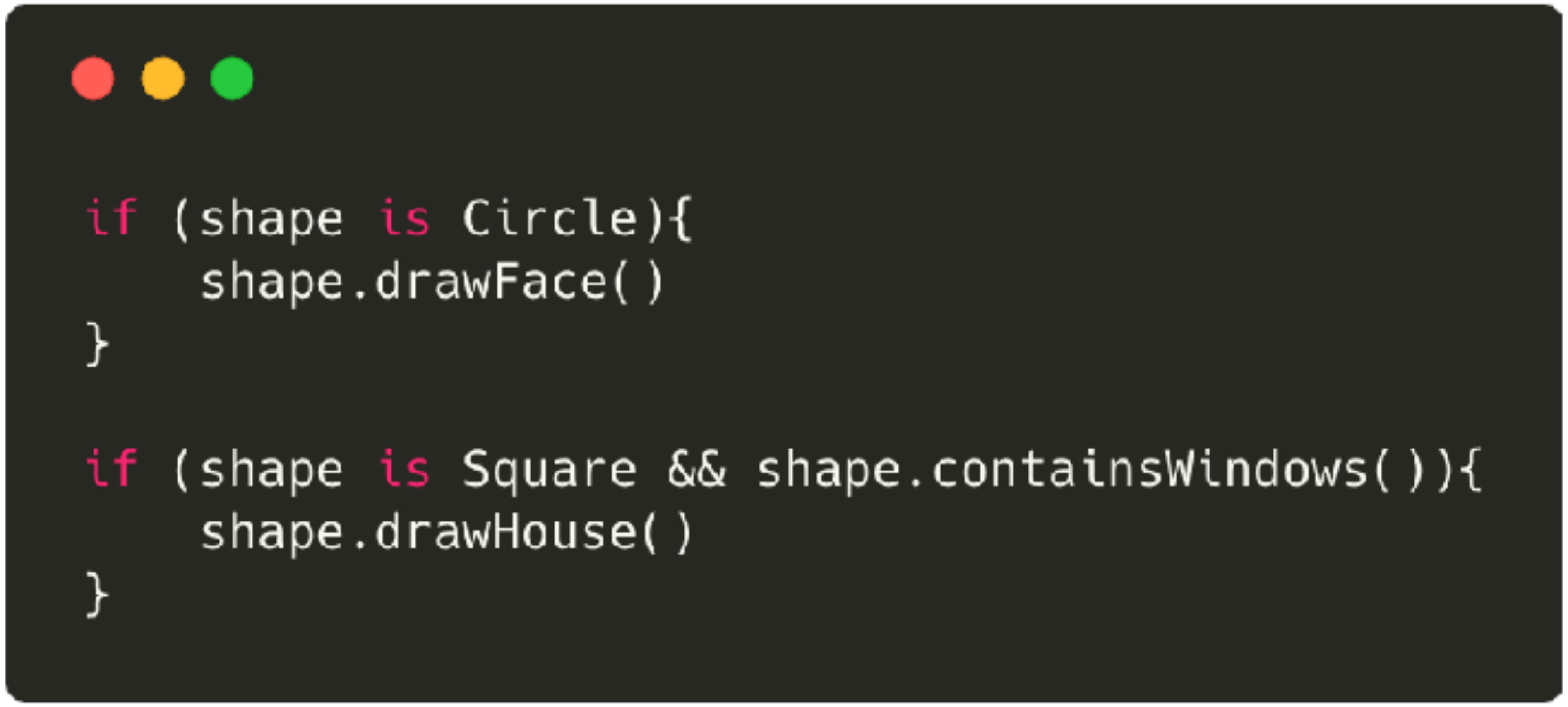
!!.



```
val len = b!!.length

//kotlin.KotlinNullPointerException
```

- Use !! If you know something will definitely not be null
- Try not to use this!!.

Smart Cast



```
if (shape is Circle){
    shape.drawFace()
}

if (shape is Square && shape.containsWindows()){
    shape.drawHouse()
}
```

- **is** keyword
- Replaces ‘instanceOf’ in Java

Koans

10min

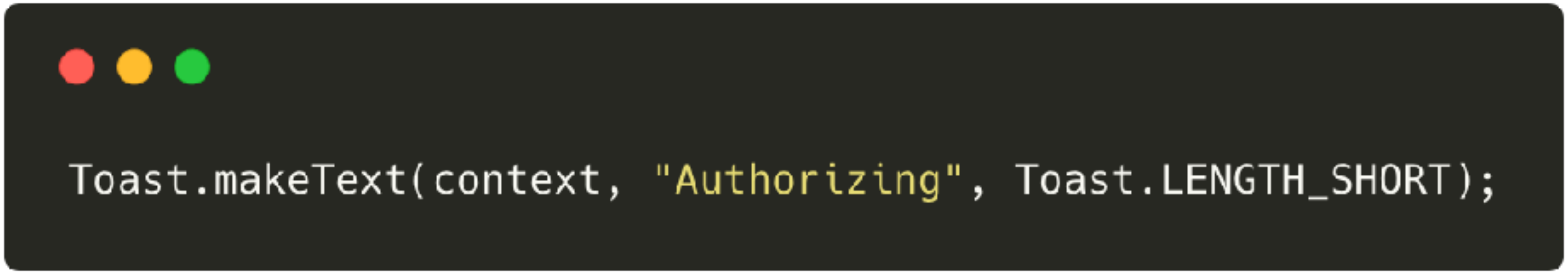
Introduction:

- Nullable types
- Smart casts

Functions

Extension Functions

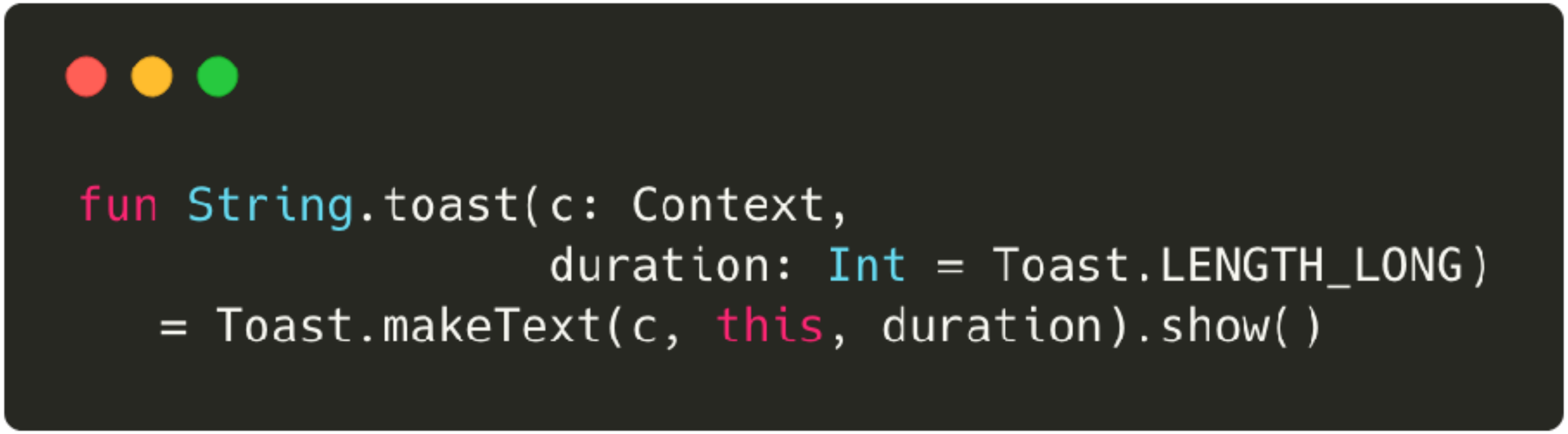
A Toast



```
Toast.makeText(context, "Authorizing", Toast.LENGTH_SHORT);
```

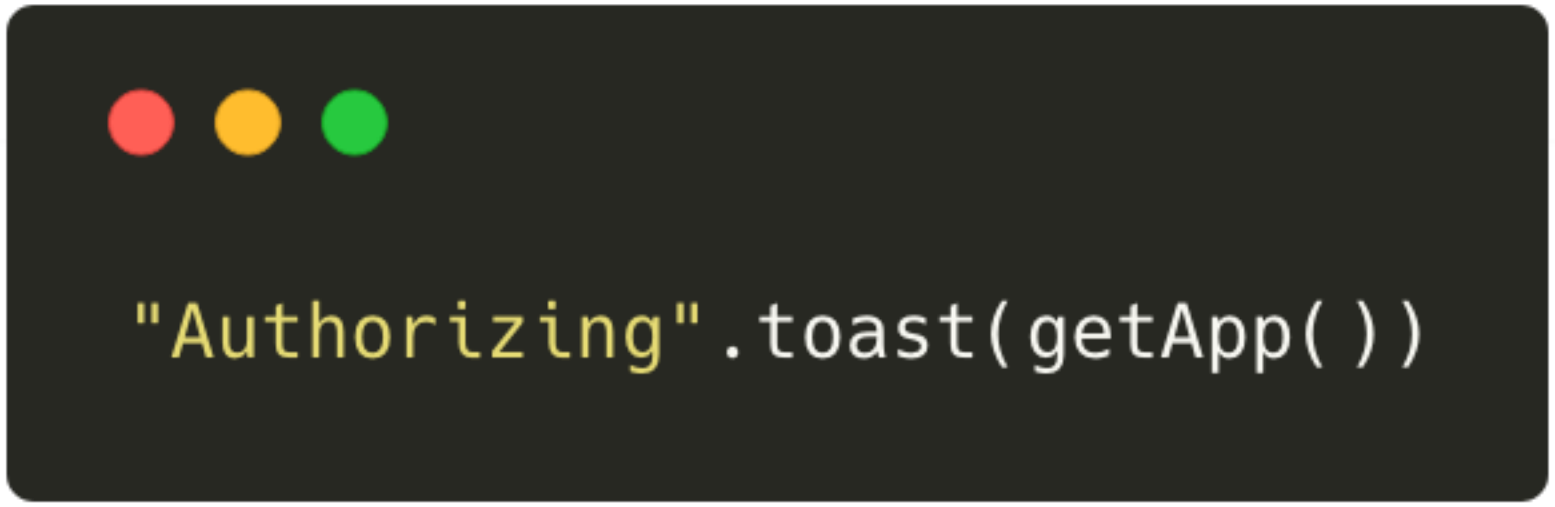
What’s wrong with this?

Kotlin Extension



```kotlin
fun String.toast(c: Context,
                 duration: Int = Toast.LENGTH_LONG)
    = Toast.makeText(c, this, duration).show()
```

Invoking

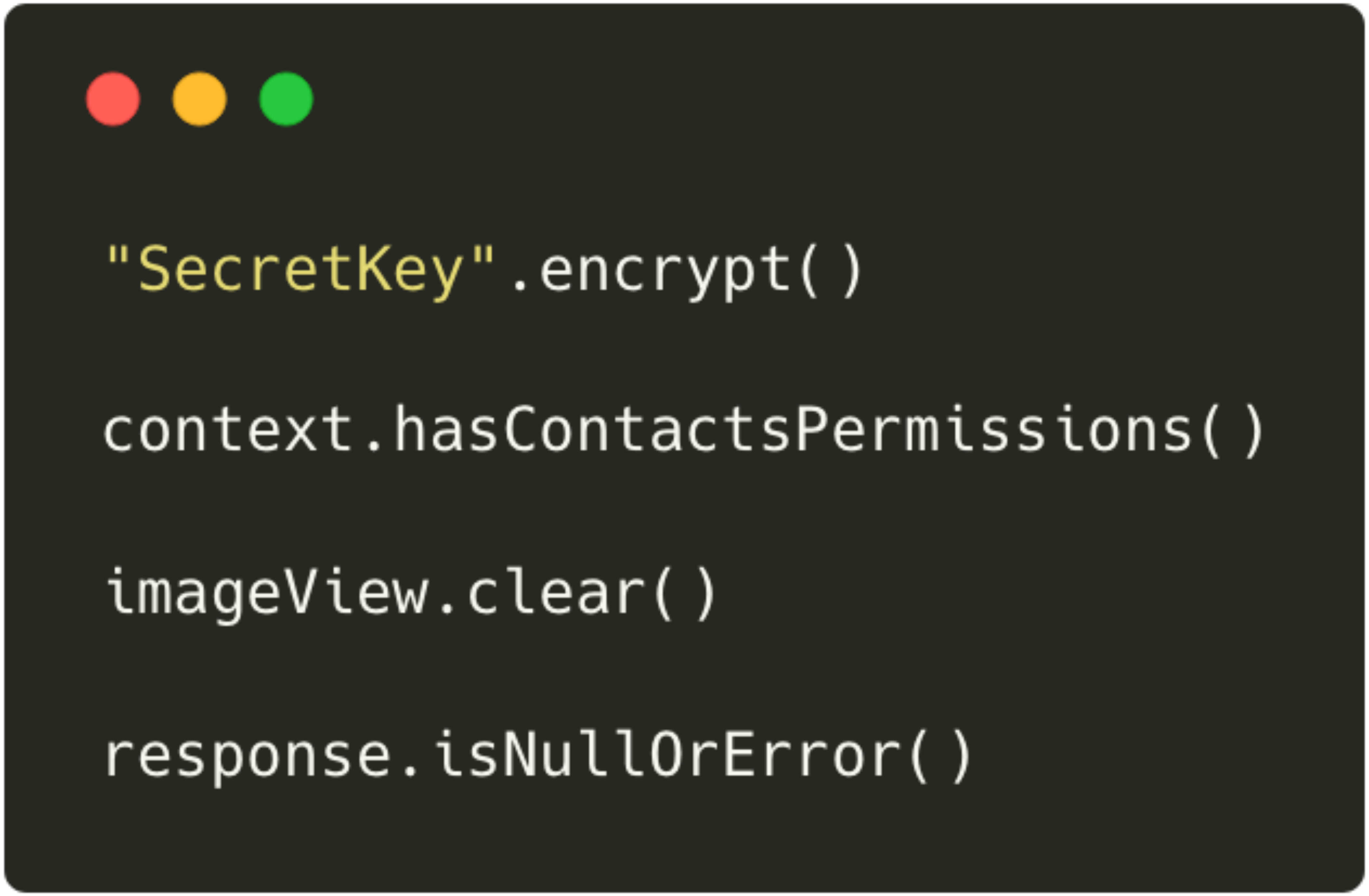


```kotlin
"Authorizing".toast(getApp())
```

Extension Function

- Adds new functions to existing classes
- Maintains the same namespace
- Colocation of functionality
- Can access protected functionality via `this`

Other Possibilities



```
"SecretKey".encrypt()

context.hasContactsPermissions()

imageView.clear()

response.isNullOrError()
```

Advantages of Extensions

- Discoverability
- Autocompletion
- Compact null checking

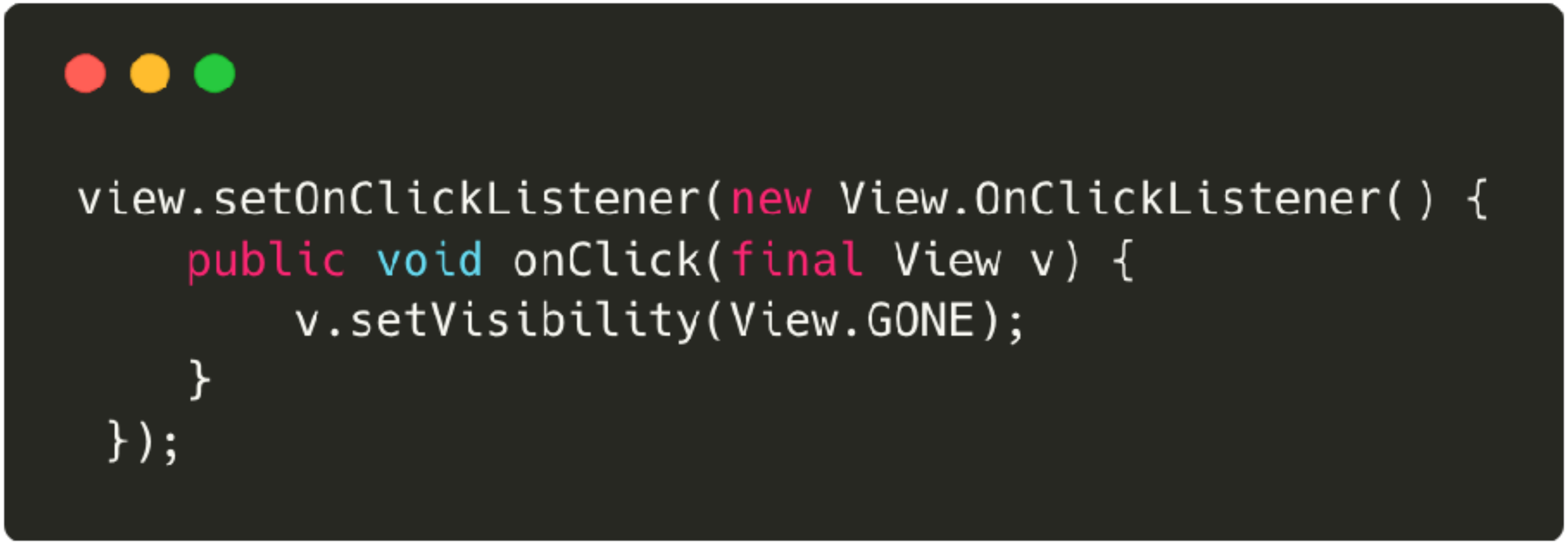
Use Cases

- Fix confusing, inconvenient, or verbose Android APIs
- Enhance 3rd party libraries without editing the source
- Create a library of helper functions for your team
- Many additions already available via Android KTX at https://github.com/android/android-ktx

Questions?

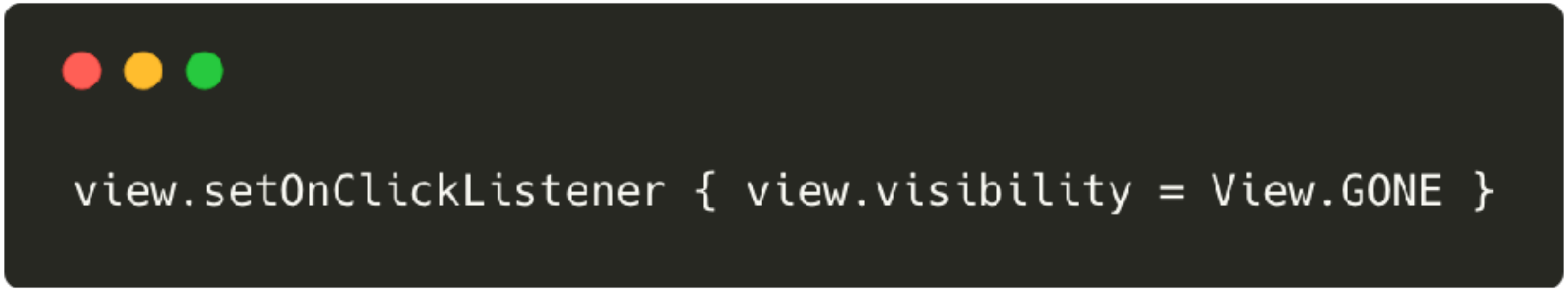
Lambdas

Traditional Interfaces



```
view.setOnClickListener(new View.OnClickListener() {
    public void onClick(final View v) {
        v.setVisibility(View.GONE);
    }
});
```

Lambda Approach



```kotlin
view.setOnClickListener { view.visibility = View.GONE }
```

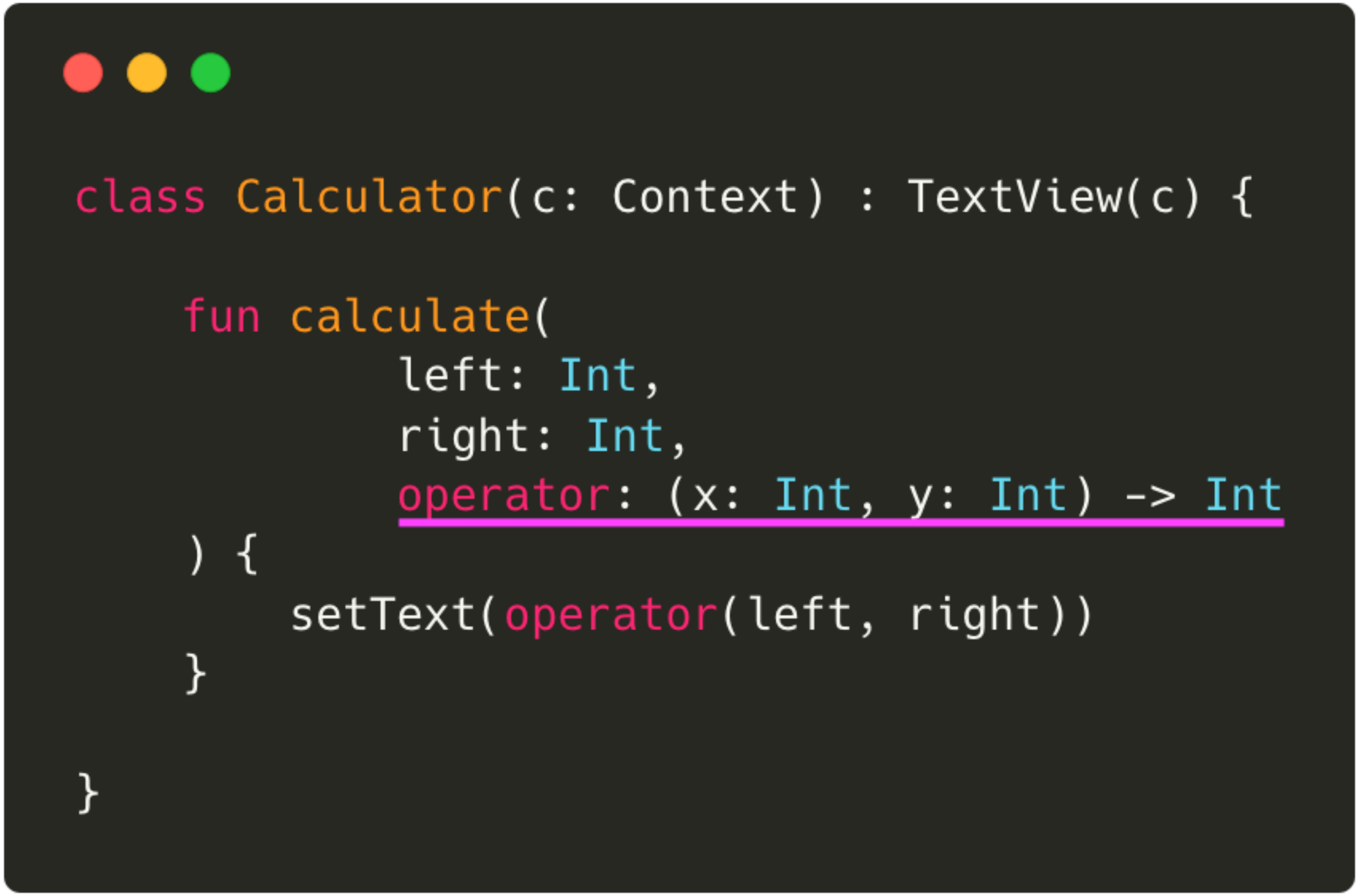
- Anonymous function
- Somewhat like an anonymous class
- Not attached to class
- A lightweight way to fulfill a contract

Higher Order Functions

A Higher Order Function Is:

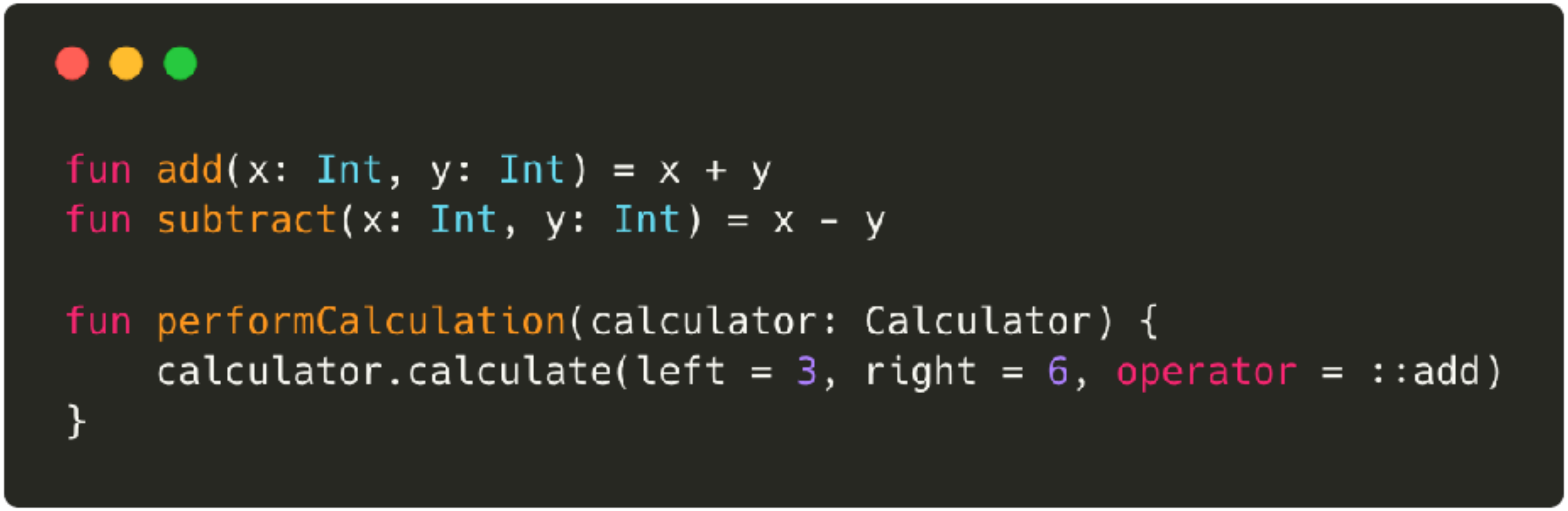
- A “first-class” citizen
- Can be passed as a parameter
- Can be stored in a variable
- Can be returned from another function

A Calculator



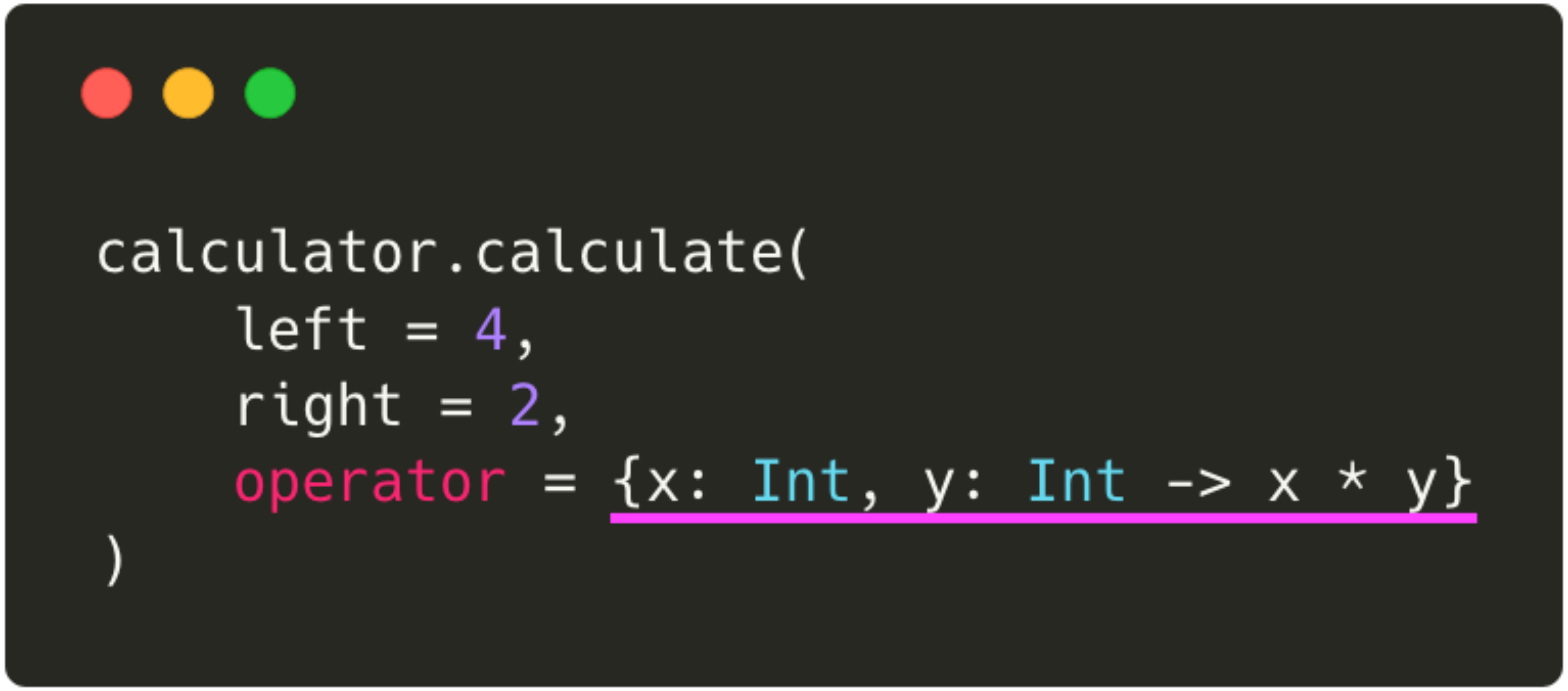
```
class Calculator(c: Context) : TextView(c) {

    fun calculate(
            left: Int,
            right: Int,
            operator: (x: Int, y: Int) -> Int
    ) {
        setText(operator(left, right))
    }

}
```

Passing Functions



```kotlin
fun add(x: Int, y: Int) = x + y
fun subtract(x: Int, y: Int) = x - y

fun performCalculation(calculator: Calculator) {
    calculator.calculate(left = 3, right = 6, operator = ::add)
}
```

Passing Lambdas



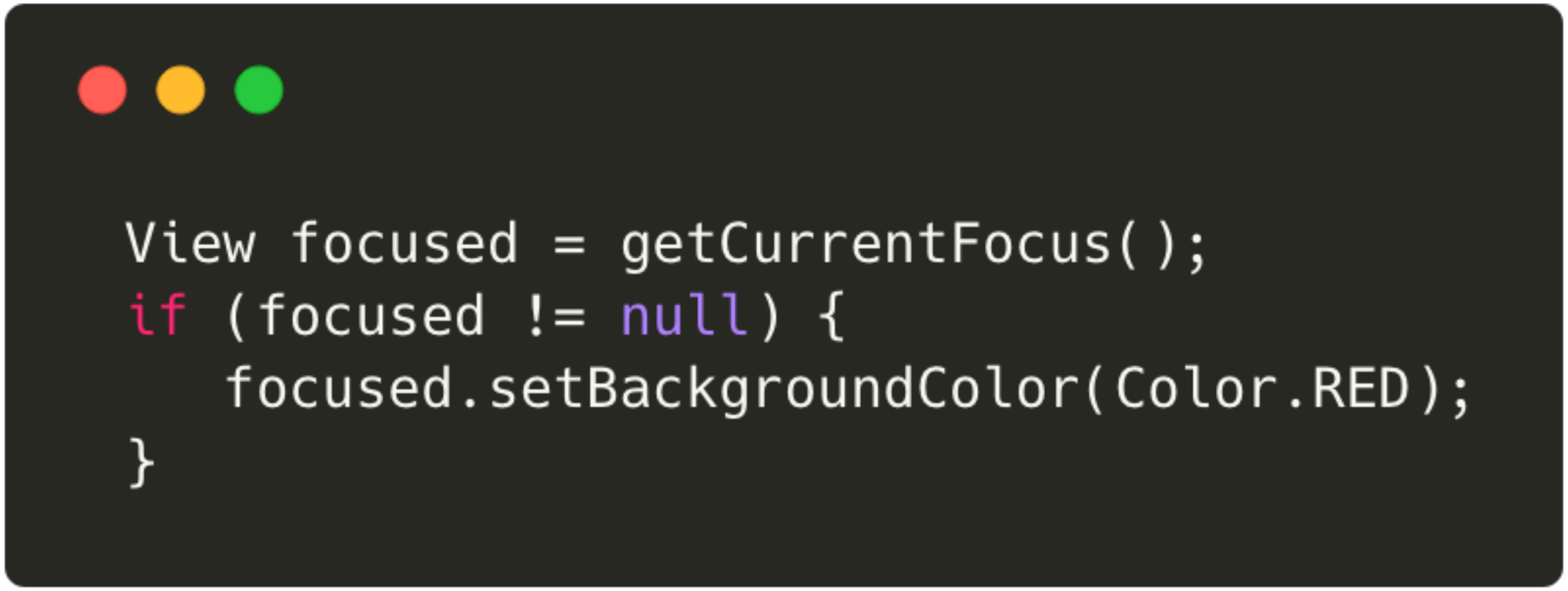
```
calculator.calculate(
    left = 4,
    right = 2,
    operator = {x: Int, y: Int -> x * y}
)
```

Questions?

Standard Higher Order Functions

Traditional Null Checking

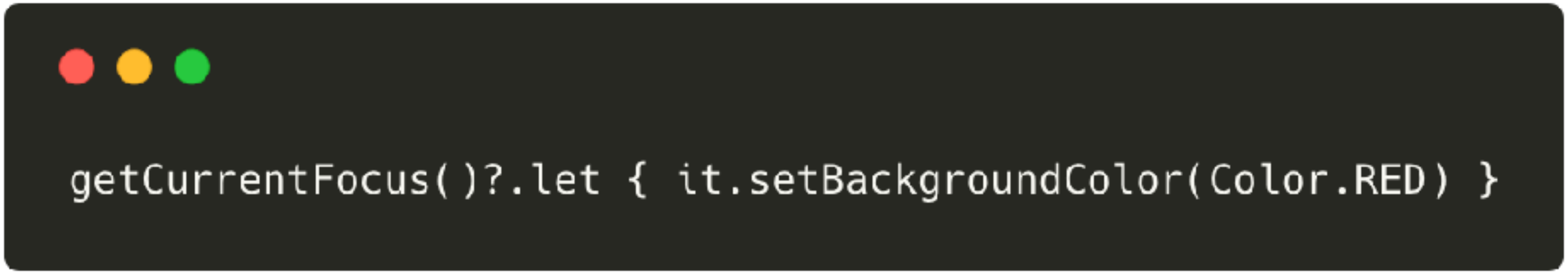
How often have you written code like this?



```
View focused = getCurrentFocus();
if (focused != null) {
    focused.setBackgroundColor(Color.RED);
}
```

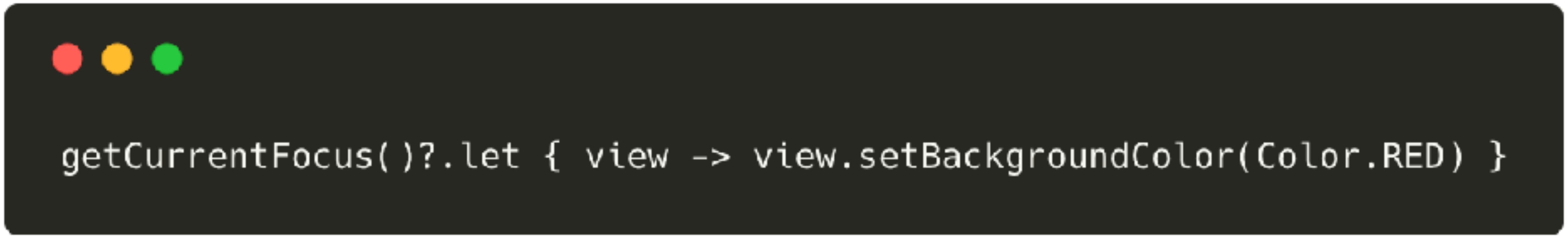
let

“let” evaluates the right side only if the left side is non-null



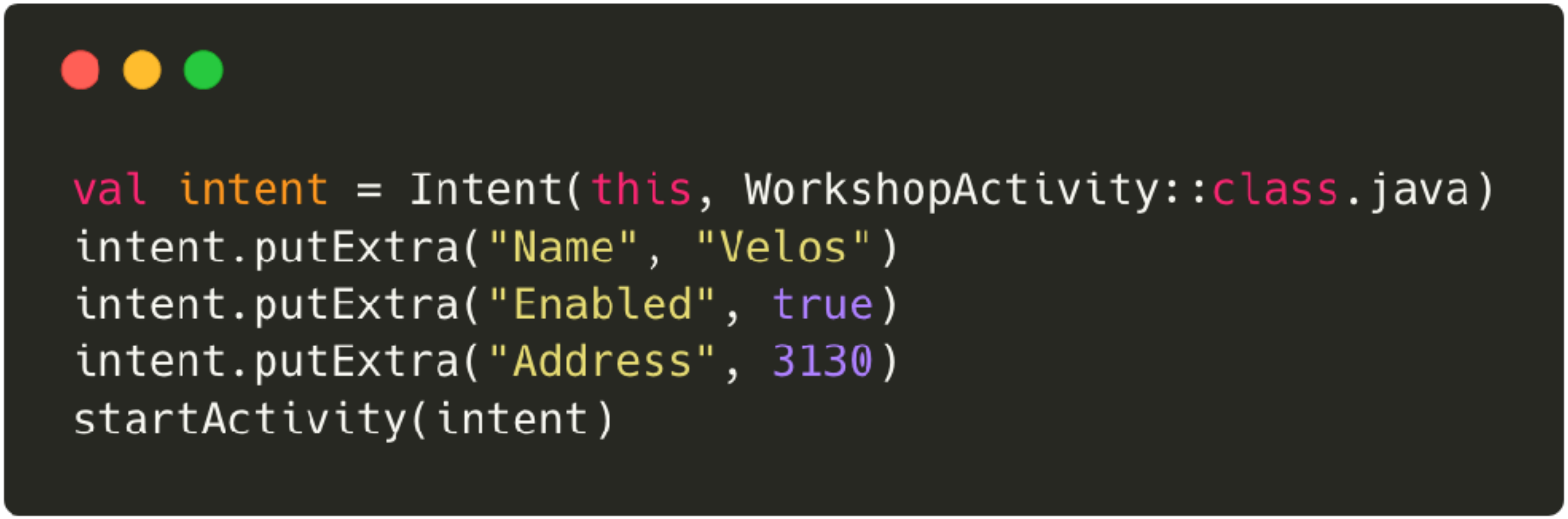
```
getCurrentFocus()?.let { it.setBackgroundColor(Color.RED) }
```

Like all lambdas, can rename parameter to be more readable



```
getCurrentFocus()?.let { view -> view.setBackgroundColor(Color.RED) }
```

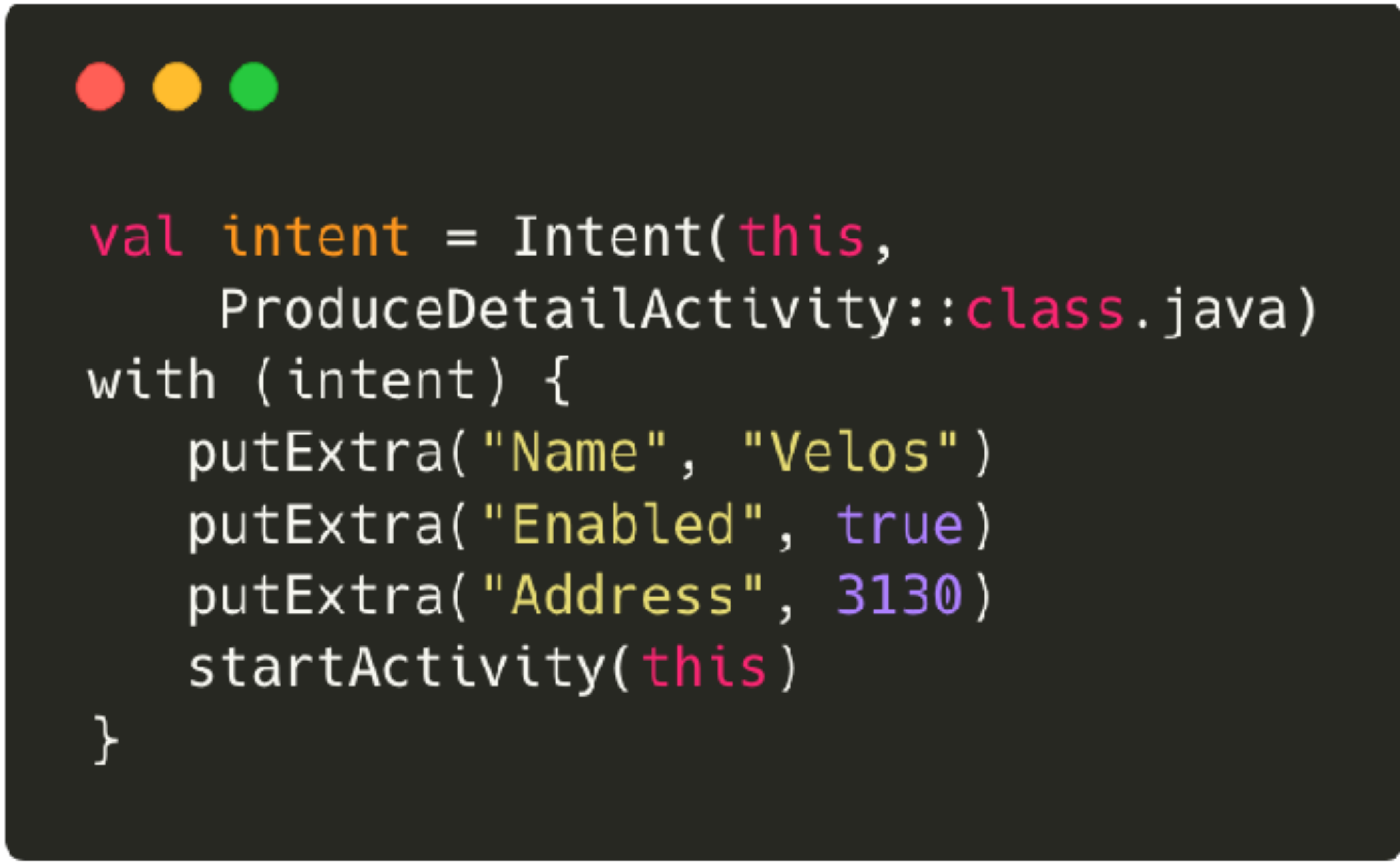
Object Configuration



```kotlin
val intent = Intent(this, WorkshopActivity::class.java)
intent.putExtra("Name", "Velos")
intent.putExtra("Enabled", true)
intent.putExtra("Address", 3130)
startActivity(intent)
```

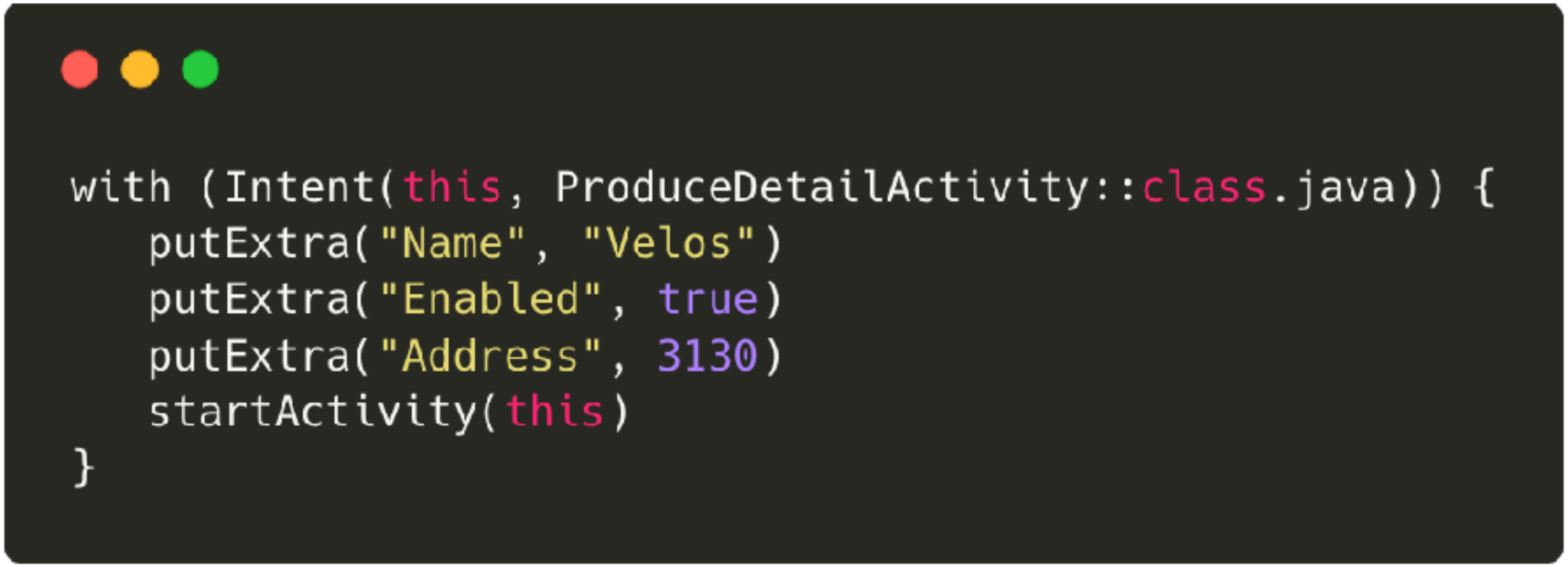
with

- Standard function that accepts a lambda
- Parameter becomes basis of the block
- Like having another “this”
- Very useful for configuring an object



```
val intent = Intent(this,
    ProduceDetailActivity::class.java)
with (intent) {
   putExtra("Name", "Velos")
   putExtra("Enabled", true)
   putExtra("Address", 3130)
   startActivity(this)
}
```

Terse Initialization

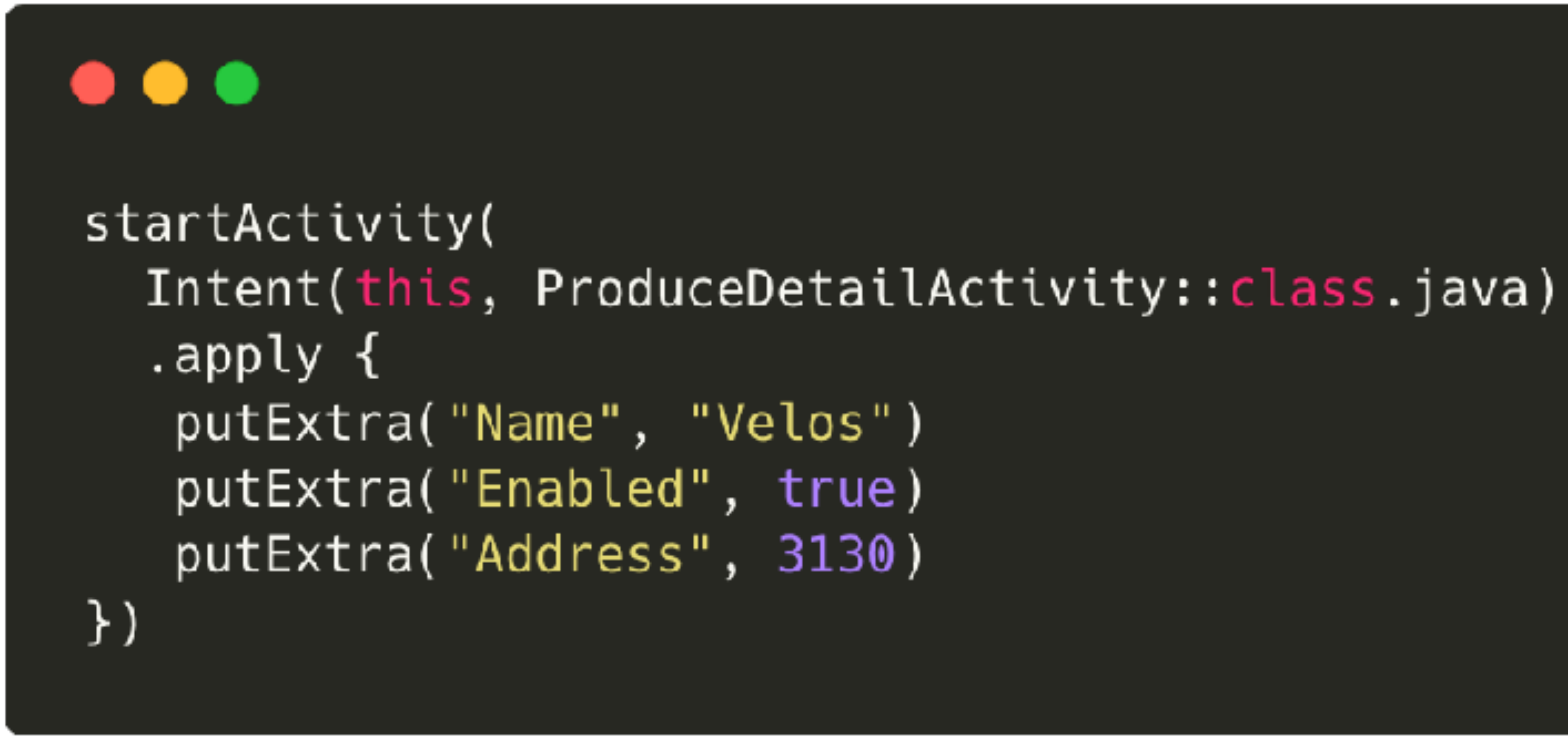


```
with (Intent(this, ProduceDetailActivity::class.java)) {
    putExtra("Name", "Velos")
    putExtra("Enabled", true)
    putExtra("Address", 3130)
    startActivity(this)
}
```

- No need to store Intent in local variable
- Can ignore it after exiting the block

apply

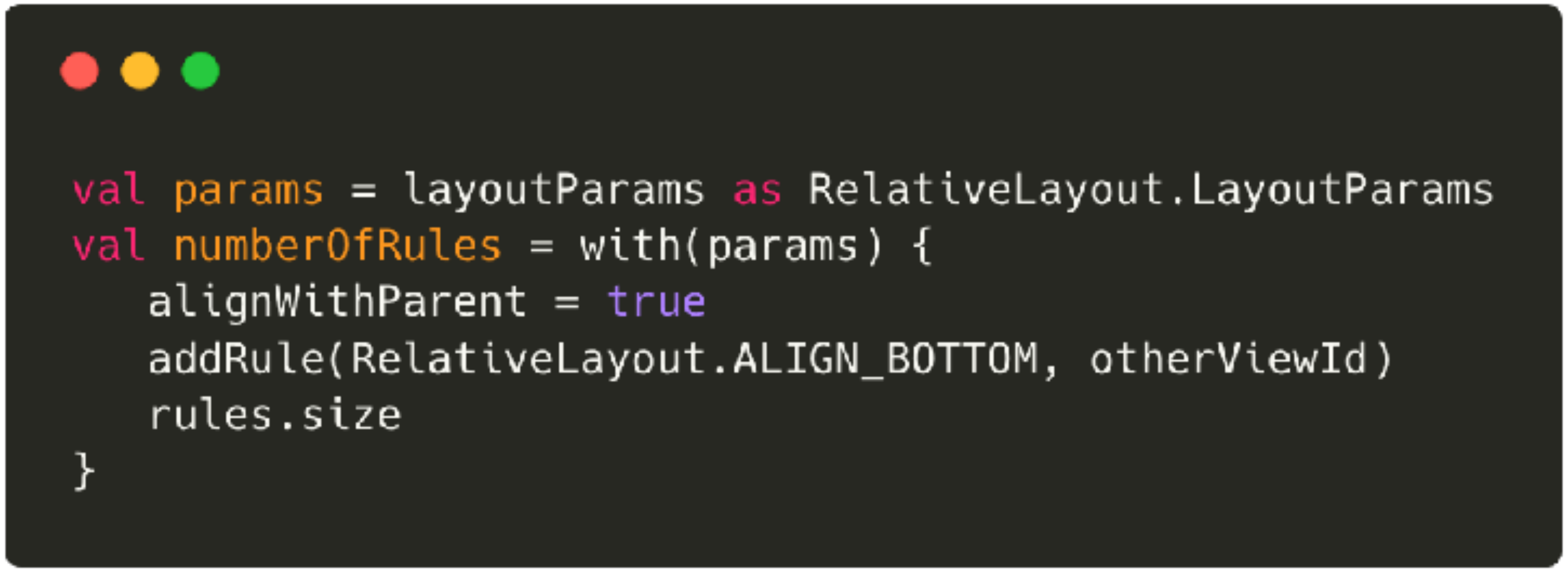
- Similar to “with”
- Extension function called on object
- Returns the object called on



```
startActivity(
  Intent(this, ProduceDetailActivity::class.java)
  .apply {
   putExtra("Name", "Velos")
   putExtra("Enabled", true)
   putExtra("Address", 3130)
})
```

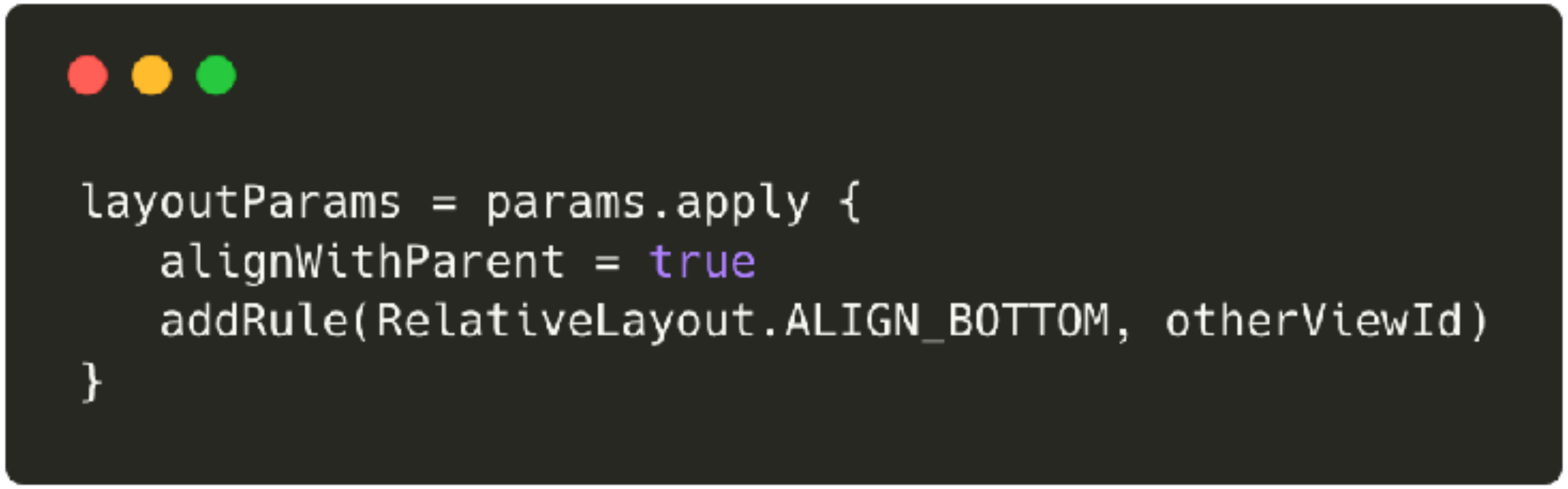
with vs. apply

“with” returns result of lambda



```
val params = layoutParams as RelativeLayout.LayoutParams
val numberOfRules = with(params) {
    alignWithParent = true
    addRule(RelativeLayout.ALIGN_BOTTOM, otherViewId)
    rules.size
}
```

“apply” returns original object



```
layoutParams = params.apply {
    alignWithParent = true
    addRule(RelativeLayout.ALIGN_BOTTOM, otherViewId)
}
```

Koans

15min

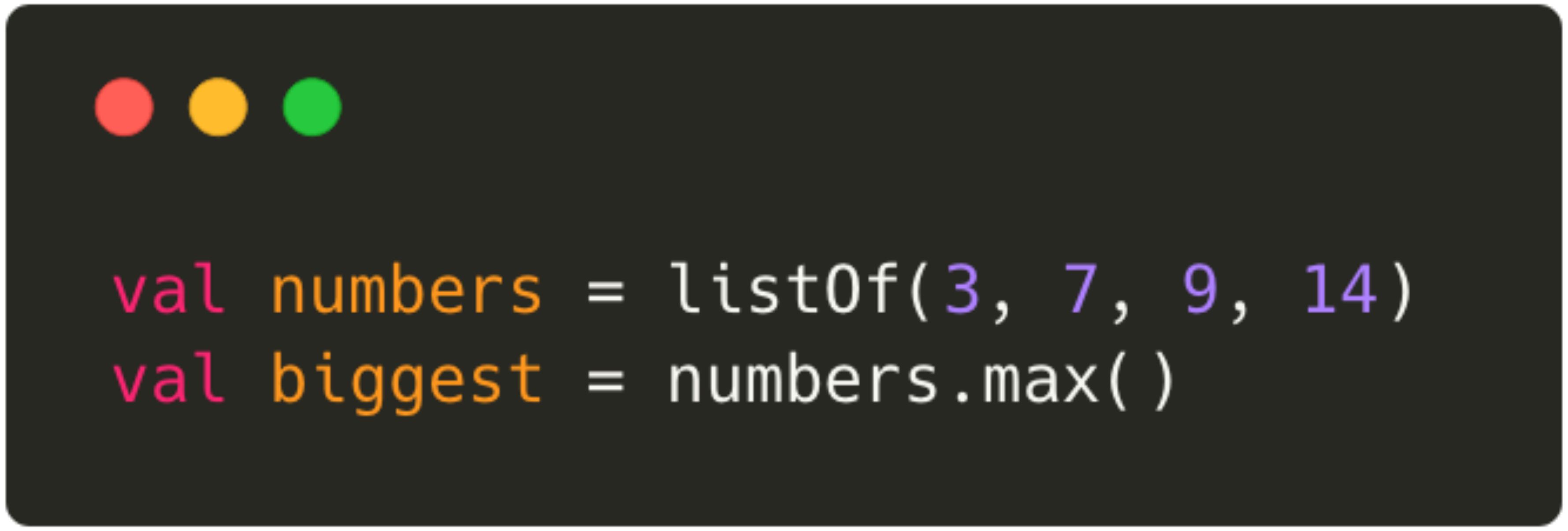
Functions:

- Lambdas
- Extension functions
- Object Expressions
- SAM conversions
- Extensions on collections

Lists

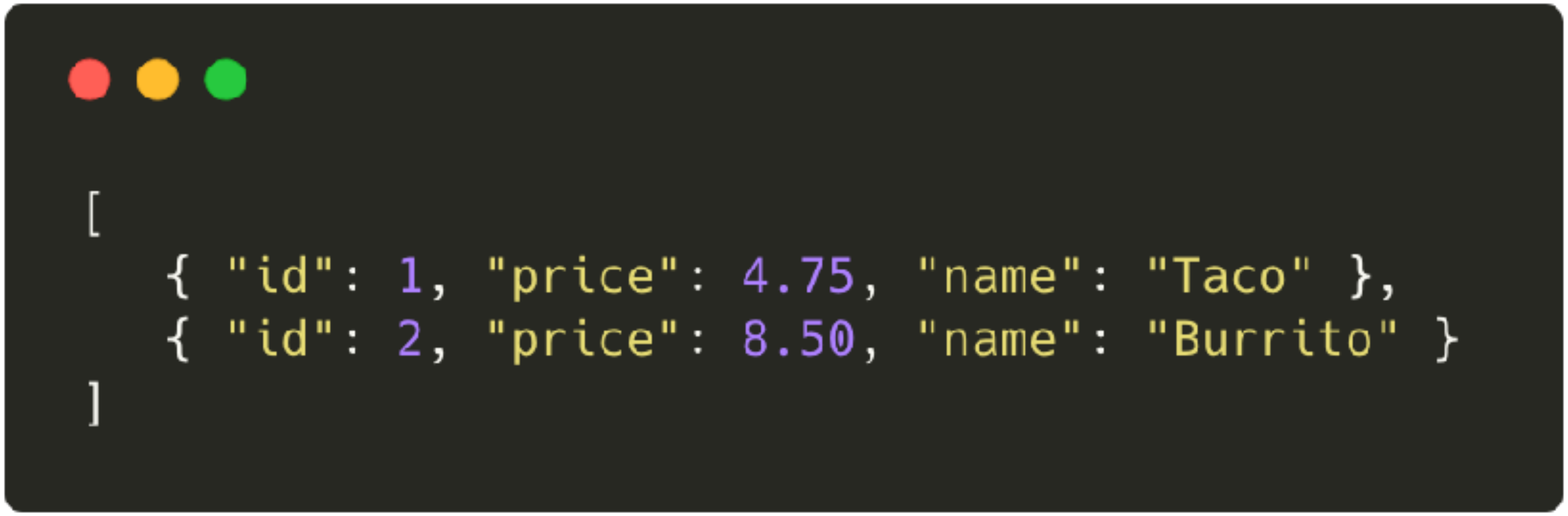
Basic List Functions

Primitive List Operations

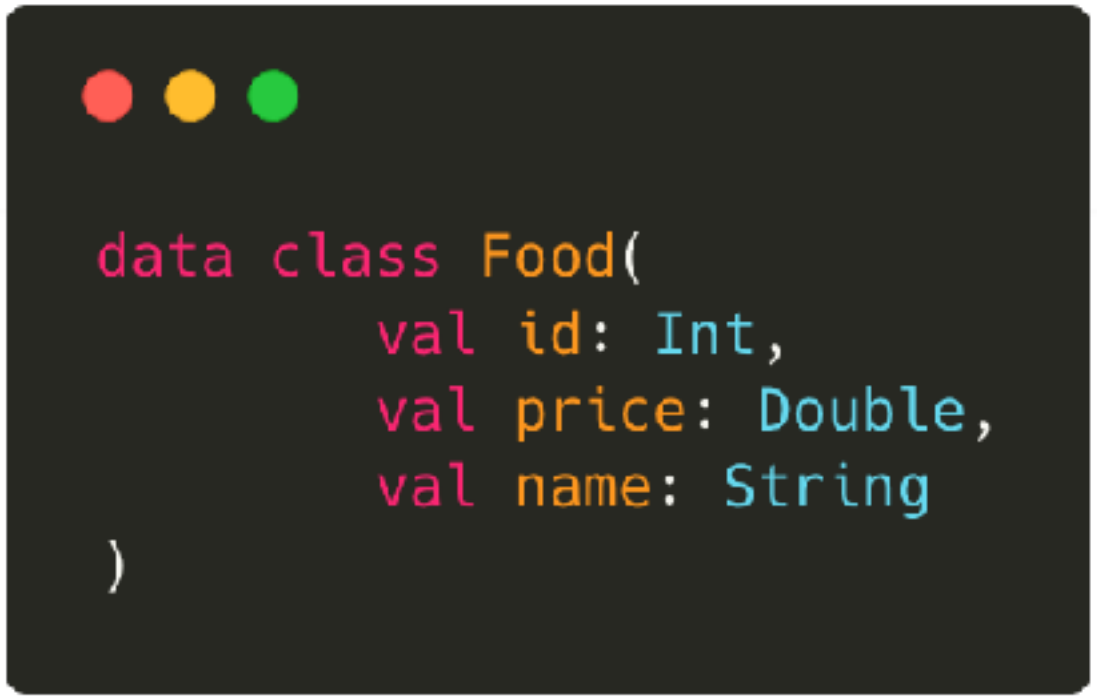


```
val numbers = listOf(3, 7, 9, 14)
val biggest = numbers.max()
```

Custom Data for List

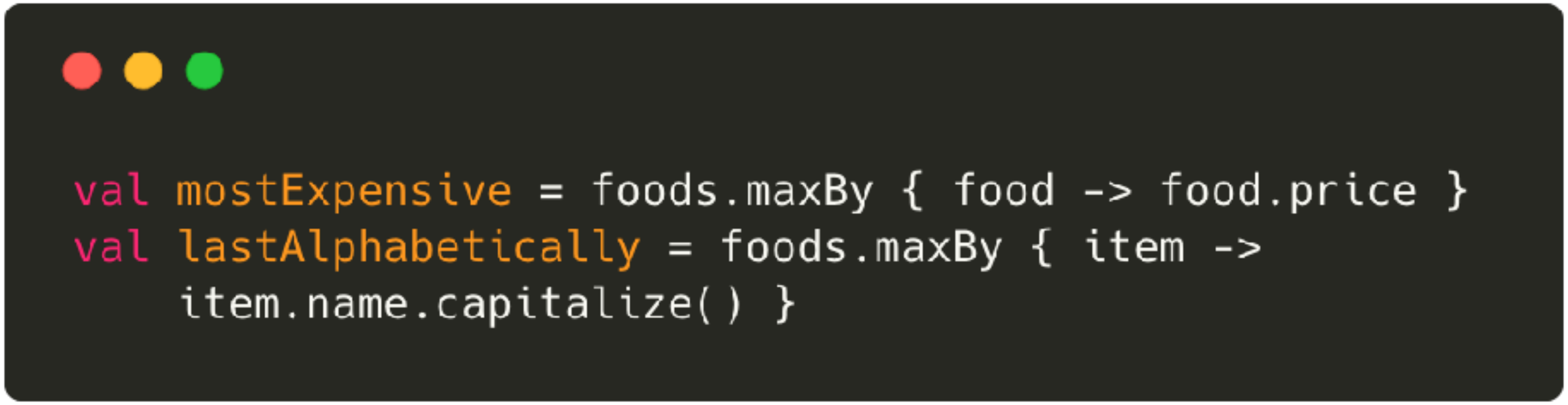


```
[
    { "id": 1, "price": 4.75, "name": "Taco" },
    { "id": 2, "price": 8.50, "name": "Burrito" }
]
```



```
data class Food(
        val id: Int,
        val price: Double,
        val name: String
)
```

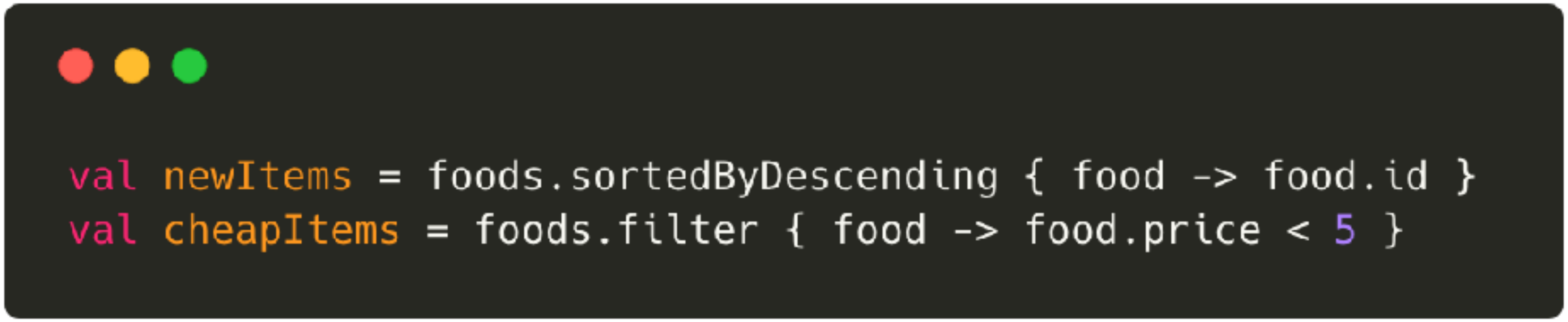
Custom List Operations



```
val mostExpensive = foods.maxBy { food -> food.price }
val lastAlphabetically = foods.maxBy { item ->
    item.name.capitalize() }
```

List Transformations

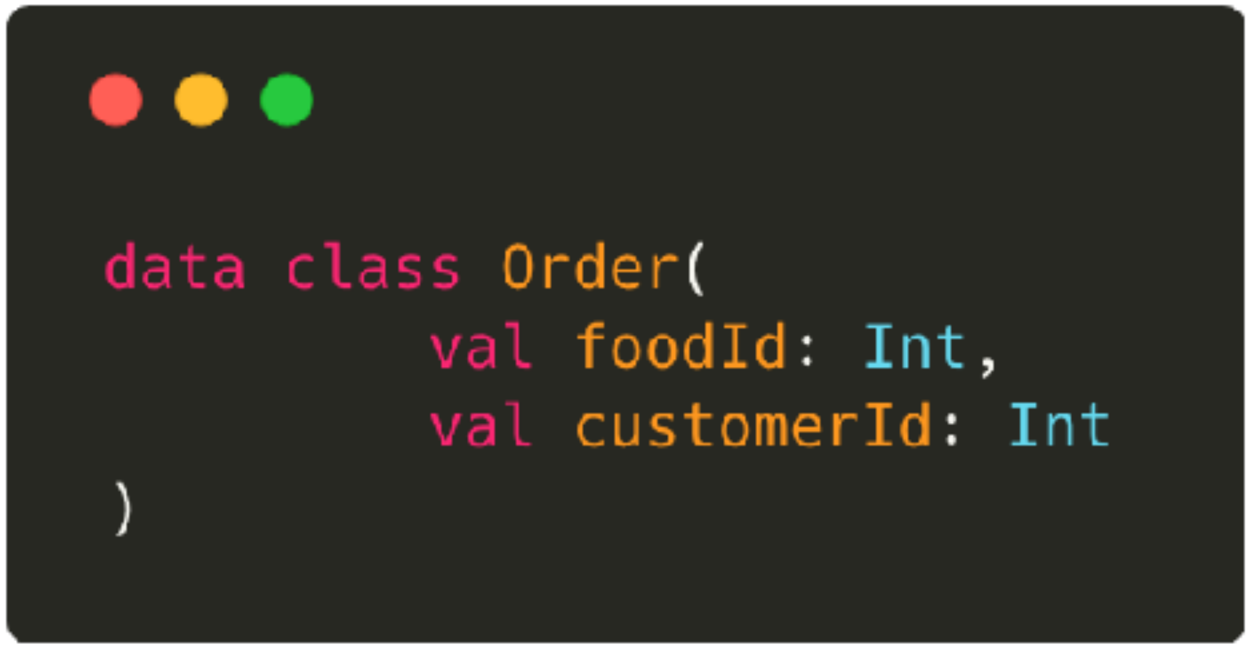
Changing a List



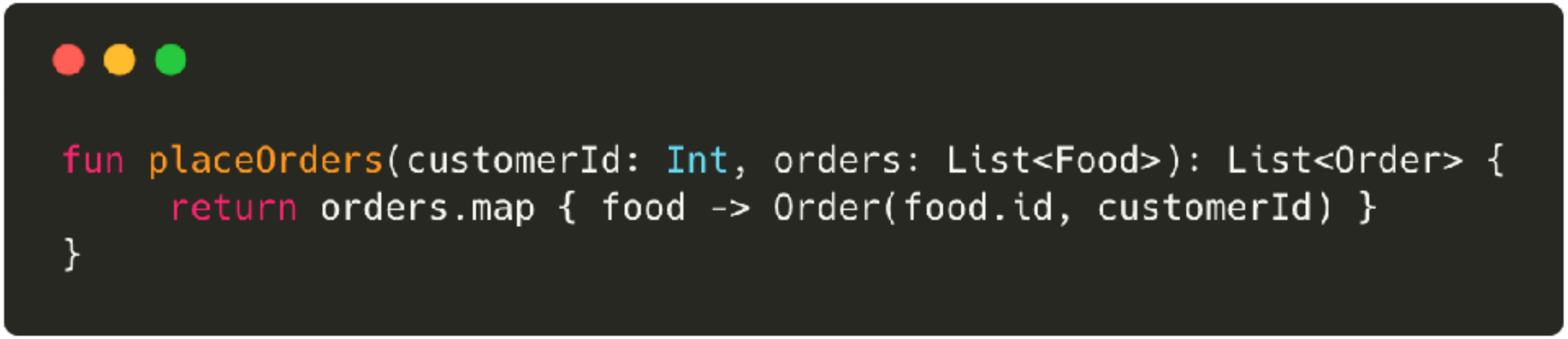
```
val newItems = foods.sortedByDescending { food -> food.id }
val cheapItems = foods.filter { food -> food.price < 5 }
```

Creates a new immutable list

Transform List To New Type



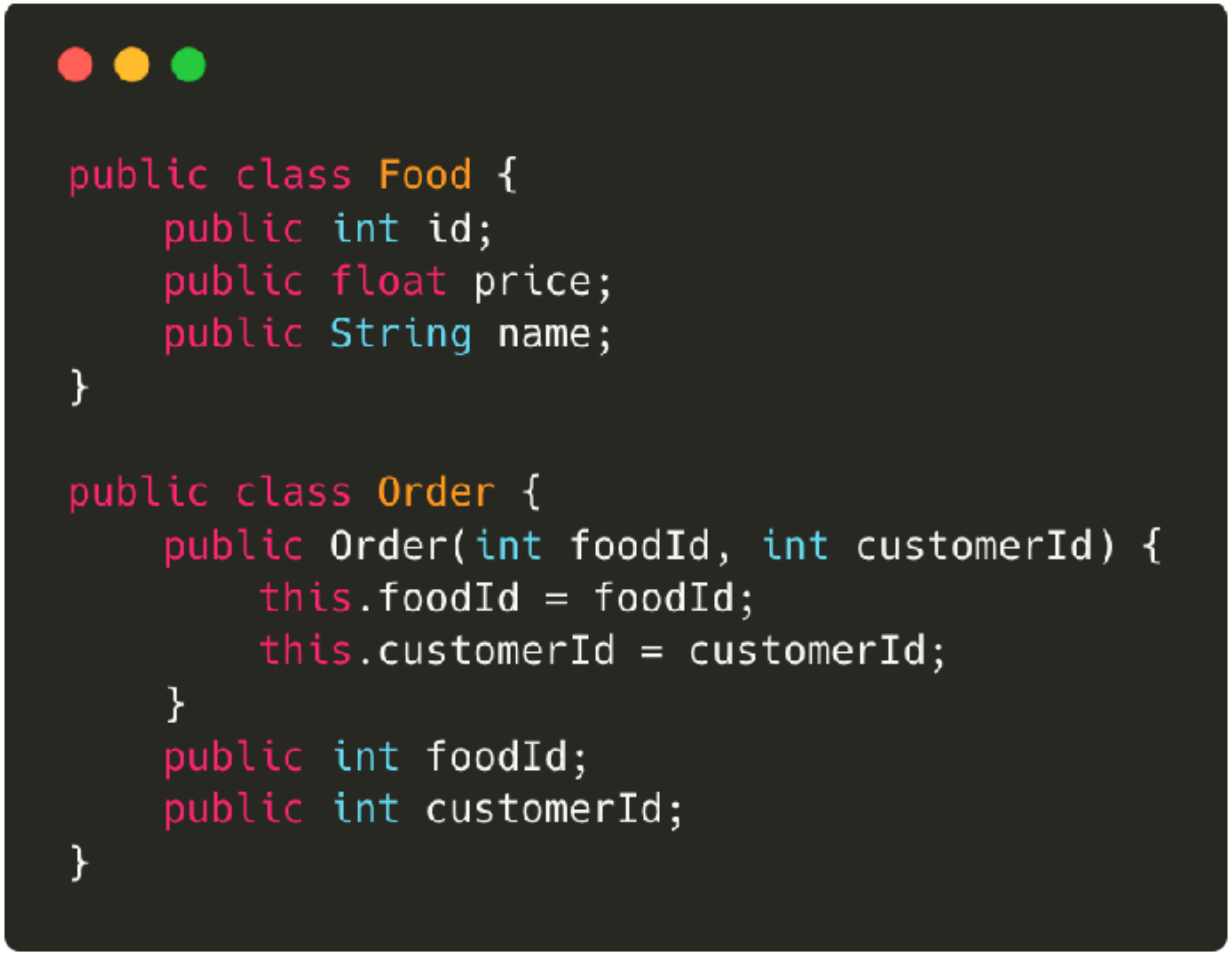
```
data class Order(
        val foodId: Int,
        val customerId: Int
)
```



```
fun placeOrders(customerId: Int, orders: List<Food>): List<Order> {
    return orders.map { food -> Order(food.id, customerId) }
}
```

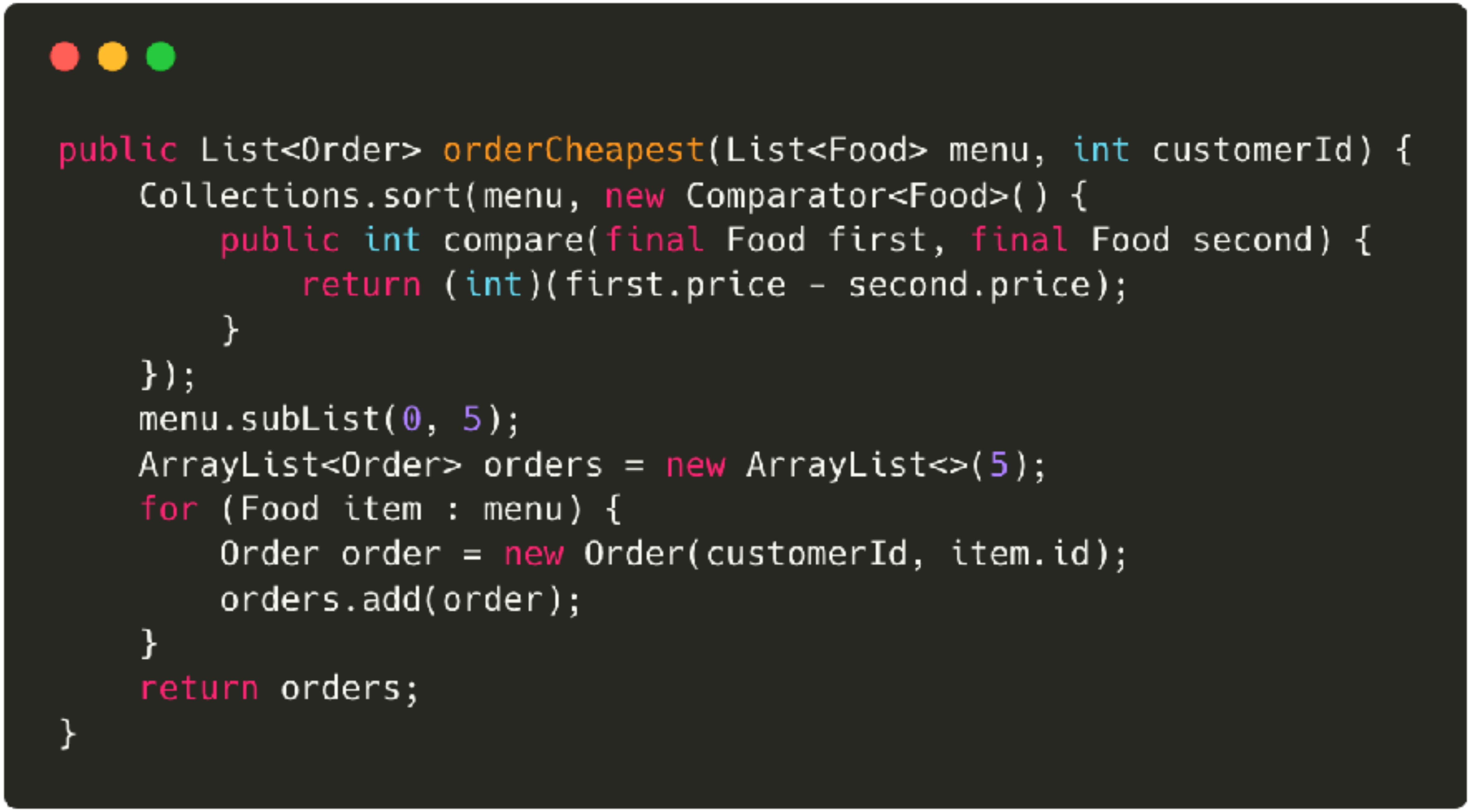
Example:
Java Refactor

Java Model Classes



```java
public class Food {
    public int id;
    public float price;
    public String name;
}

public class Order {
    public Order(int foodId, int customerId) {
        this.foodId = foodId;
        this.customerId = customerId;
    }
    public int foodId;
    public int customerId;
}
```

Initial Implementation



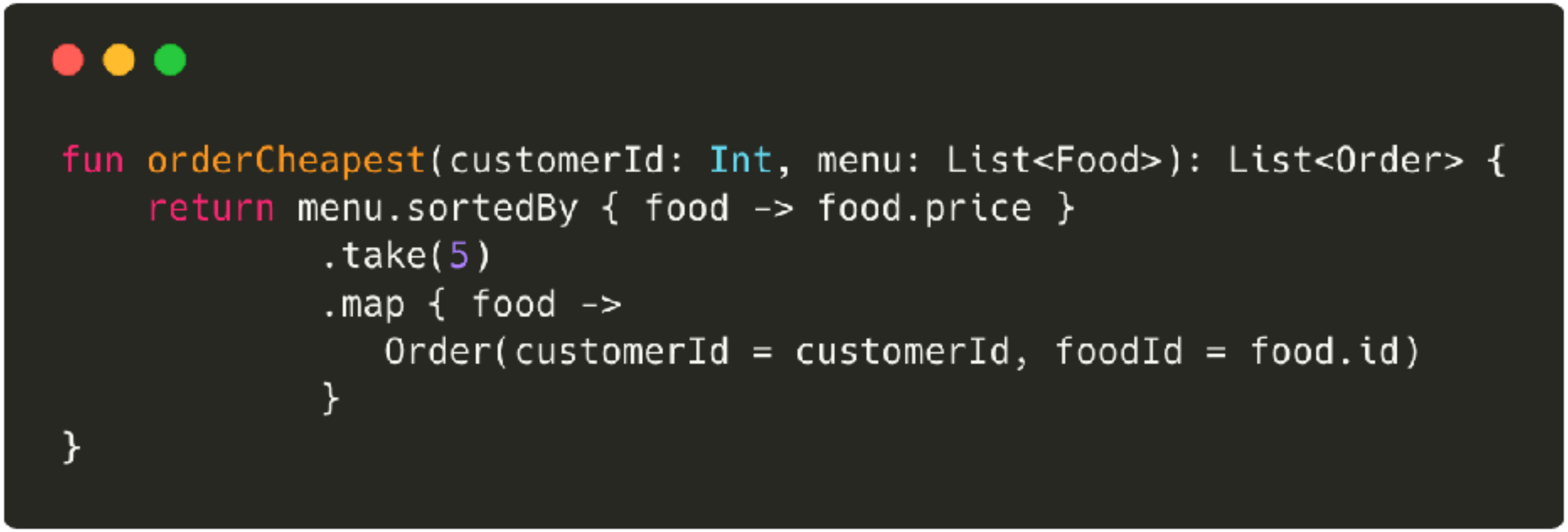
```
public List<Order> orderCheapest(List<Food> menu, int customerId) {
    Collections.sort(menu, new Comparator<Food>() {
        public int compare(final Food first, final Food second) {
            return (int)(first.price - second.price);
        }
    });
    menu.subList(0, 5);
    ArrayList<Order> orders = new ArrayList<>(5);
    for (Food item : menu) {
        Order order = new Order(customerId, item.id);
        orders.add(order);
    }
    return orders;
}
```

Can you spot the bugs?

Remaining Issues

- Side effects to provided list
- Code bloat
- Mixed APIs
- Requires study

Kotlin Implementation



```kotlin
fun orderCheapest(customerId: Int, menu: List<Food>): List<Order> {
    return menu.sortedBy { food -> food.price }
            .take(5)
            .map { food ->
               Order(customerId = customerId, foodId = food.id)
            }
}
```

Koans

15min

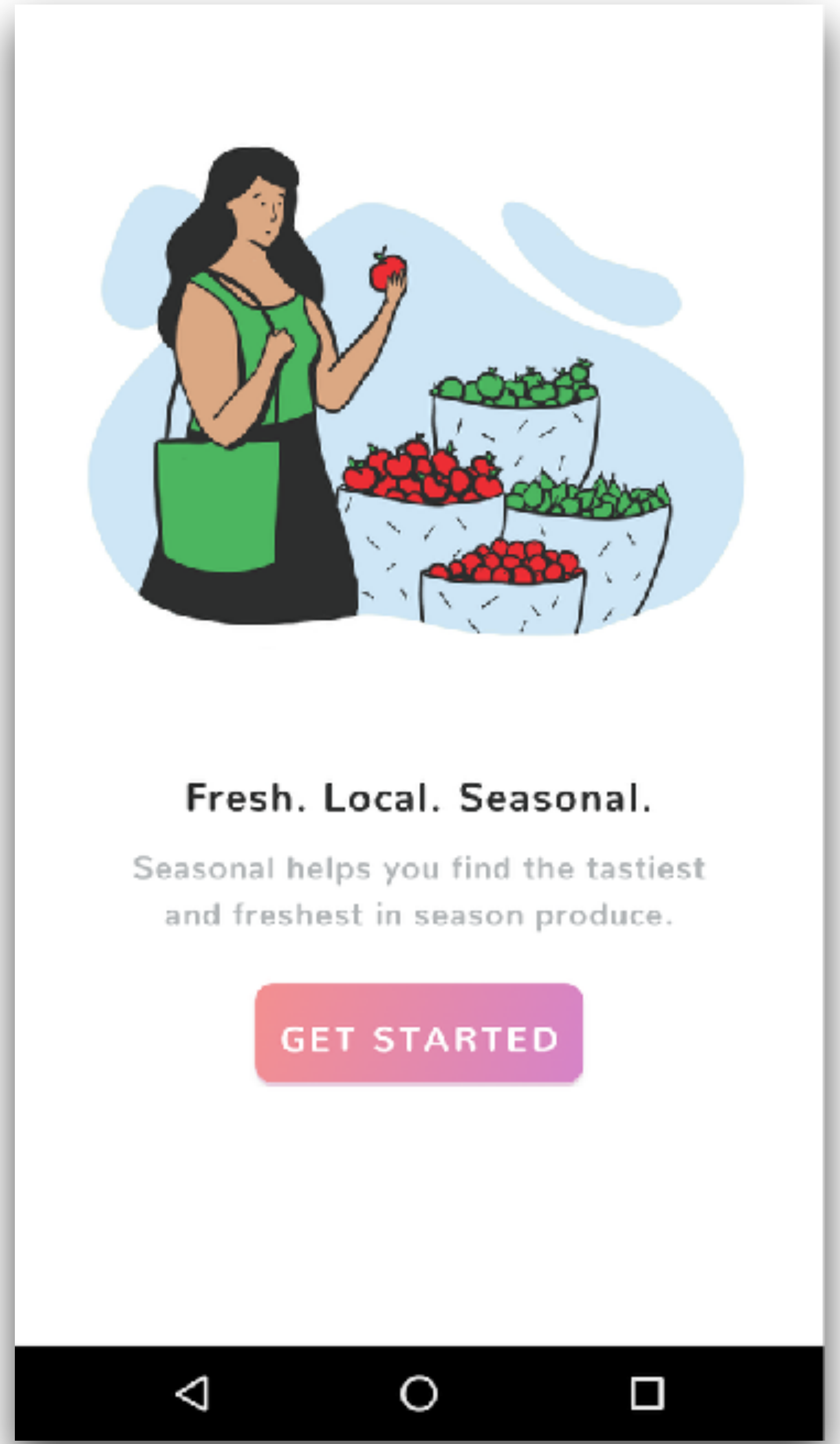
Collections:

- Introduction
- Filter map
- Flat map
- (Bonus) Max min
- Sort
- (Bonus) GroupBy
- Get used to new style

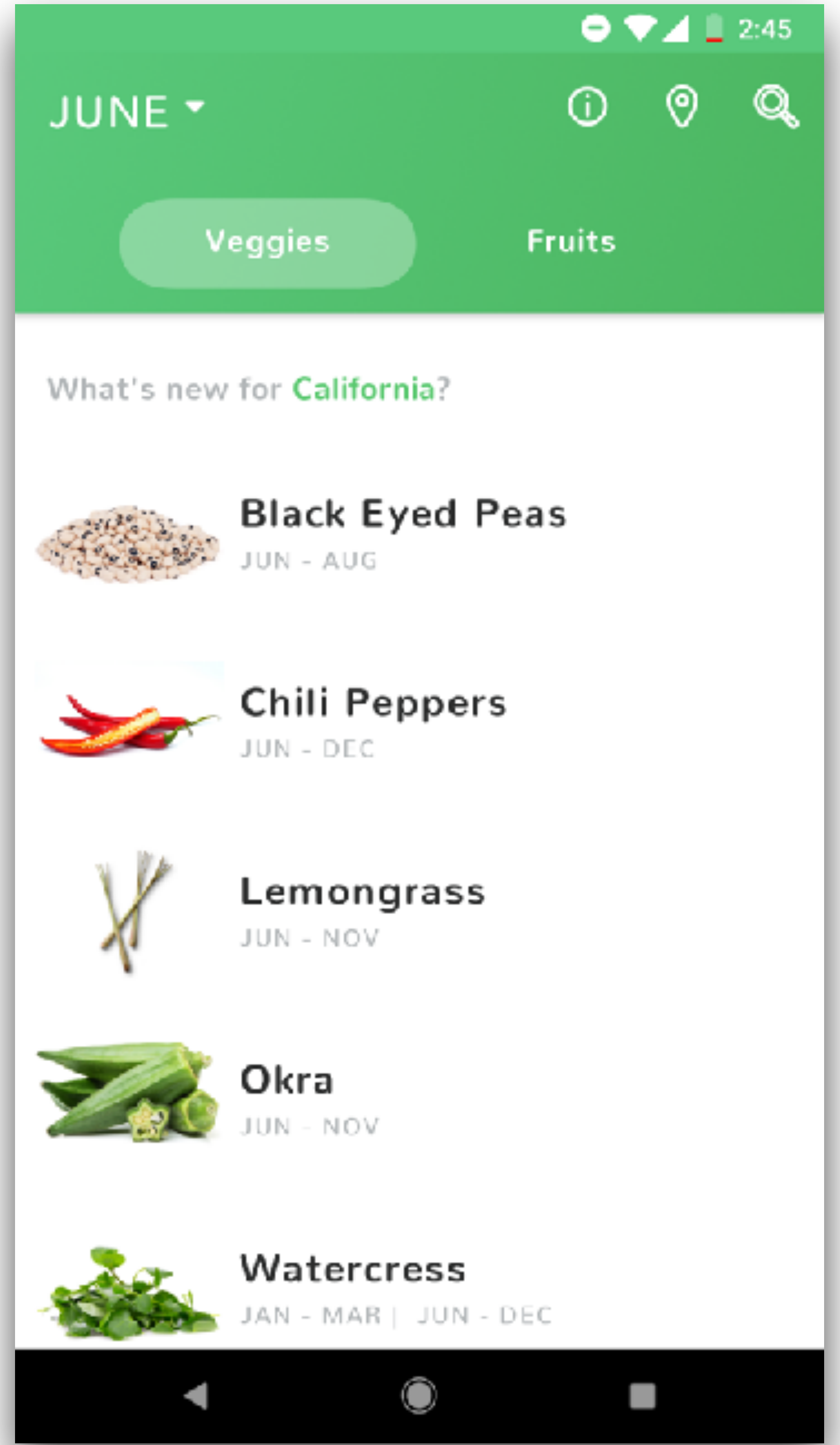
Kotlin for Android

Seasonal

100% Kotlin


Fresh. Local. Seasonal.

Seasonal helps you find the tastiest and freshest in season produce.



2:45

JUNE

Veggies

Fruits

What's new for California?

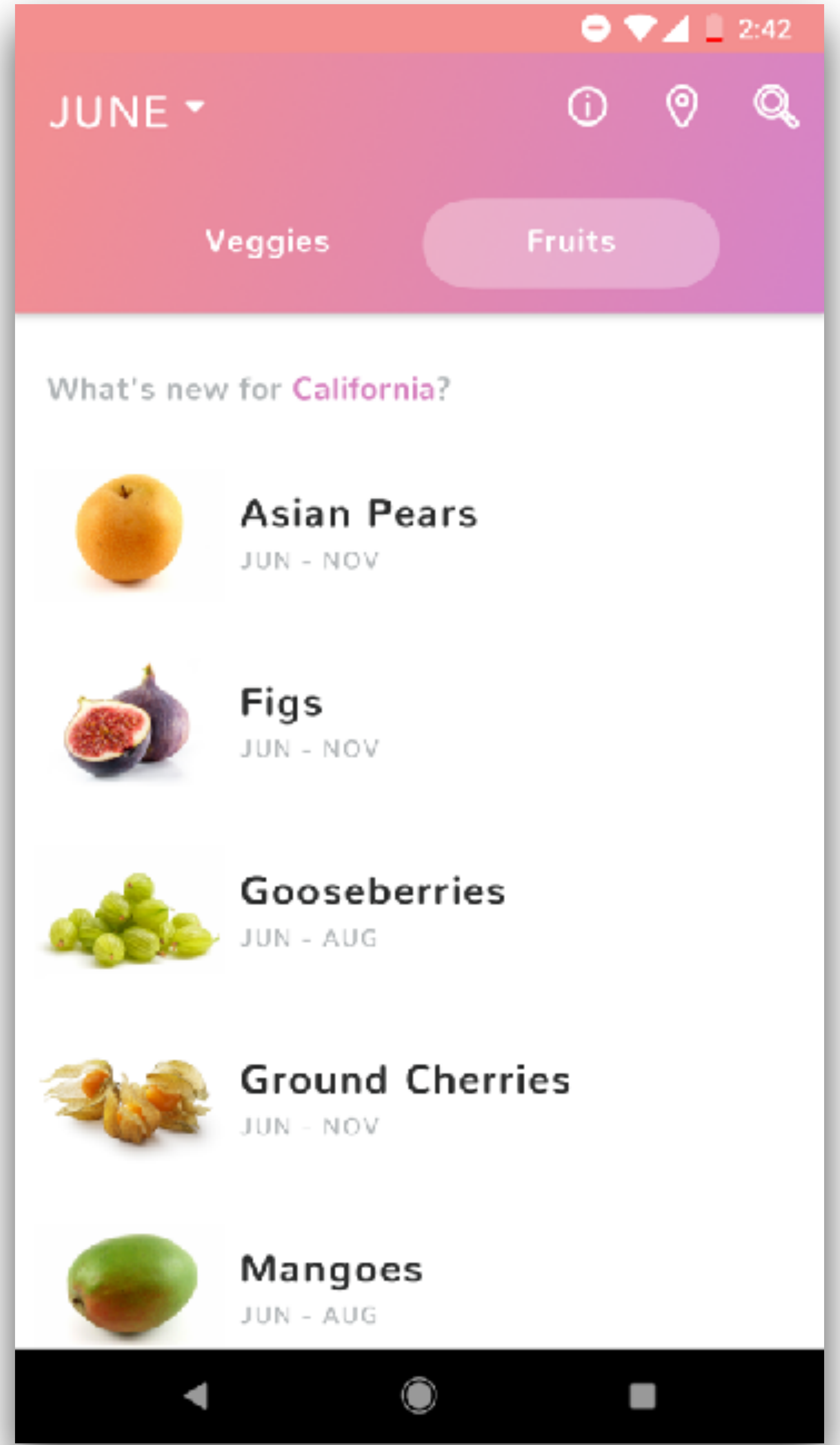
Black Eyed Peas
JUN - AUG

Chili Peppers
JUN - DEC

Lemongrass
JUN - NOV

Okra
JUN - NOV

Watercress
JAN - MAR | JUN - DEC



2:42

JUNE

Veggies

Fruits

What's new for California?

Asian Pears
JUN - NOV

Figs
JUN - NOV

Gooseberries
JUN - AUG

Ground Cherries
JUN - NOV

Mangoes
JUN - AUG

Task 0: Setup

- Create New Project (SeasonalClone)
- Include Kotlin Support.
- API 21+
- BasicActivity template
- Run project -> “Hello World!”

github.com/velos/SeasonalClone/wiki

Recap: Day 1

- Variables and Functions
- Immutability
- Data Classes
- Hierarchies
- Nullability
- Extension Functions
- Lambdas/Higher Order functions
- Collections: Filters and Maps
- Android Task 1