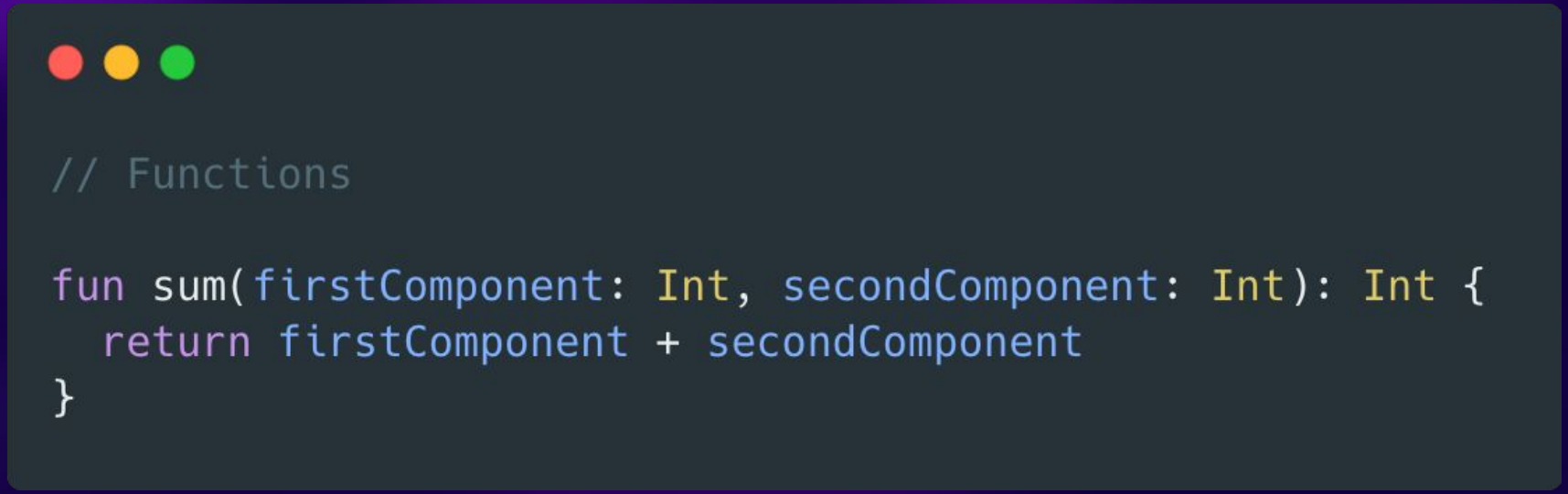
# Desarrollo Android

## Clase 03

# Funciones

# Funciones



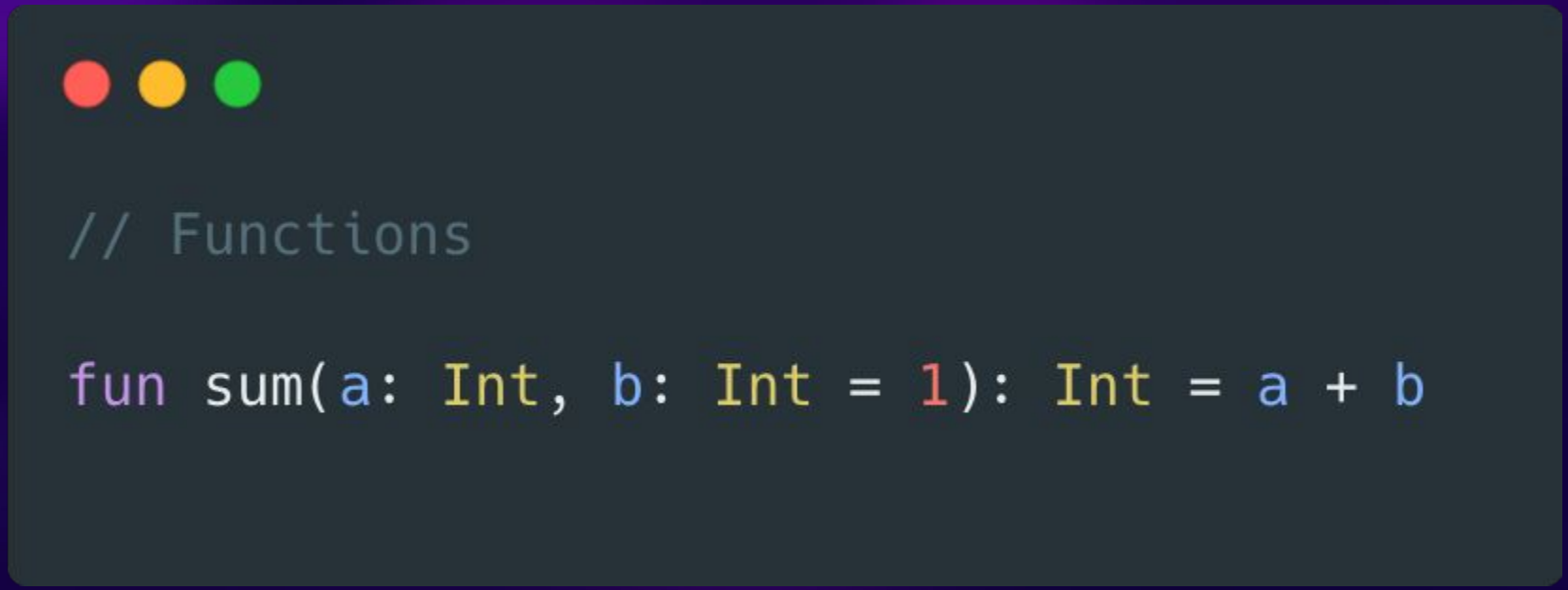
```
// Functions

fun sum(firstComponent: Int, secondComponent: Int): Int {
  return firstComponent + secondComponent
}
```

# Default parameters

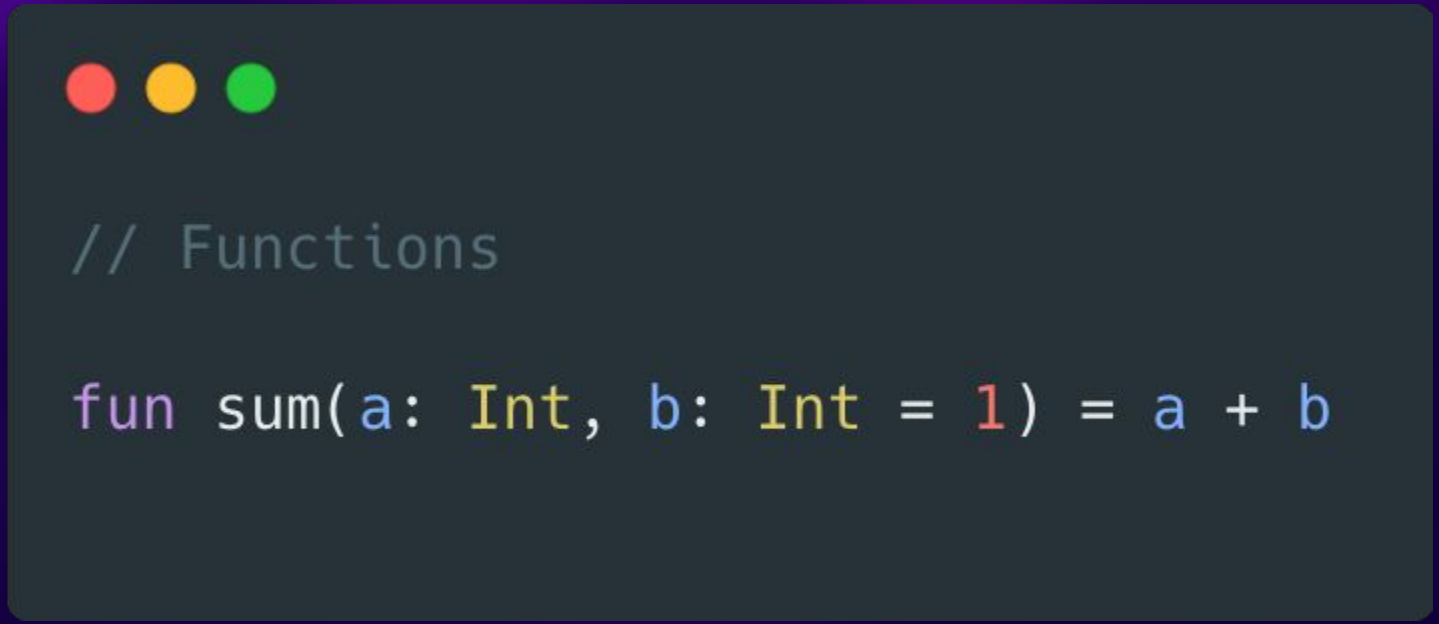
VALOR POR DEFECTO

```
// Functions

fun sum(firstComponent: Int, secondComponent: Int = 1): Int {
  return firstComponent + secondComponent
}

...

sum(1, 0)
sum(1)
```

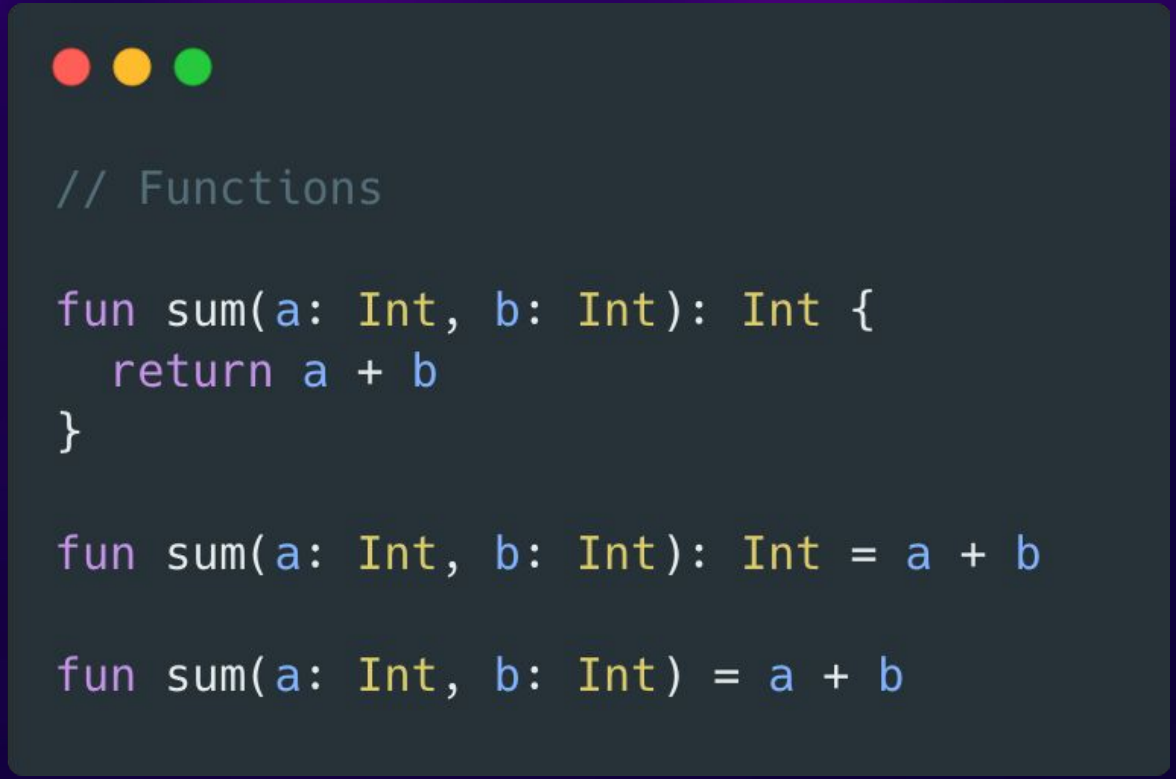
# Single-expression functions



# Type inference



# Funciones



```
// Functions

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

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

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

# Extension functions

```
// Extension functions

fun appendExclamationMark(text: String): String {
  return "$text!"
}

fun appendExclamationMark(text: String) = "$text!"

...

appendExclamationMark("Hello world") //"Hello world!"
```

es una forma de extender el texto.

# Extension functions

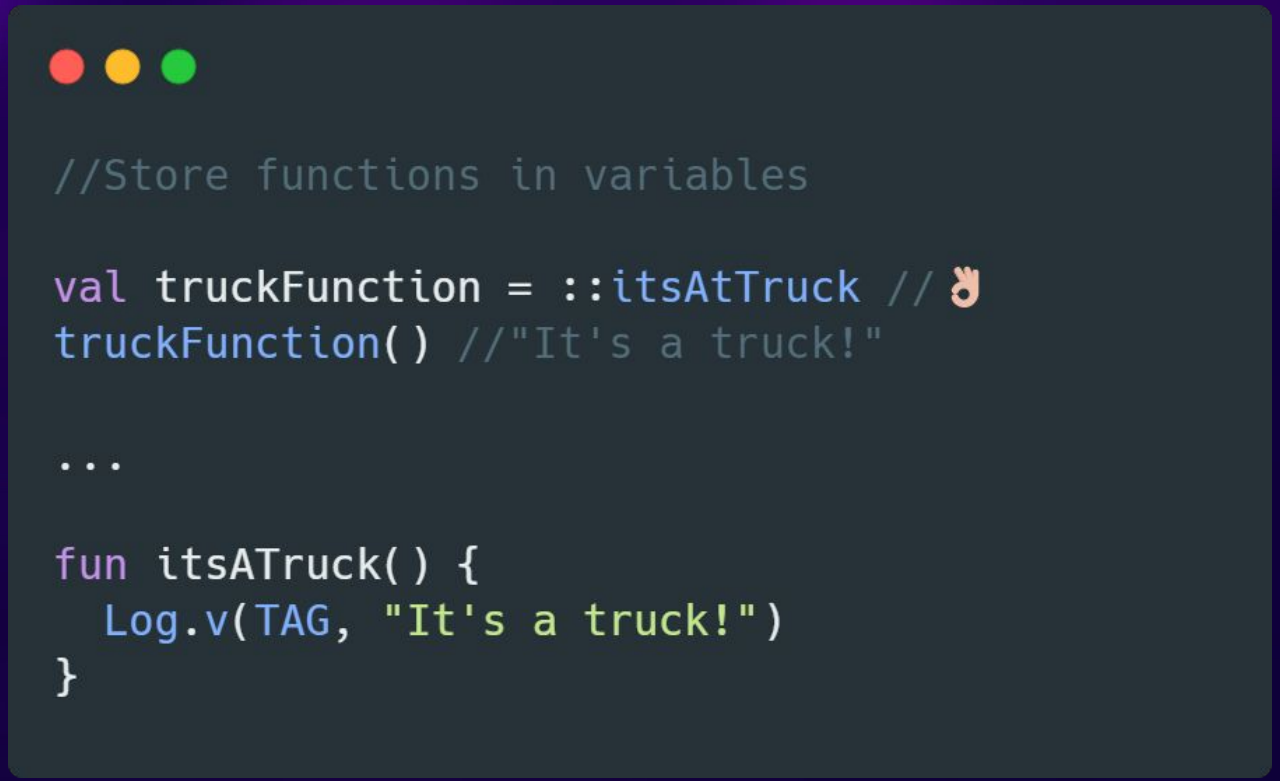
```
// Extension functions

fun String.appendExclamationMark() = "$this!"

"Hello world".appendExclamationMark() //"Hello world!"
```

Se agrega una extencion a la funcion

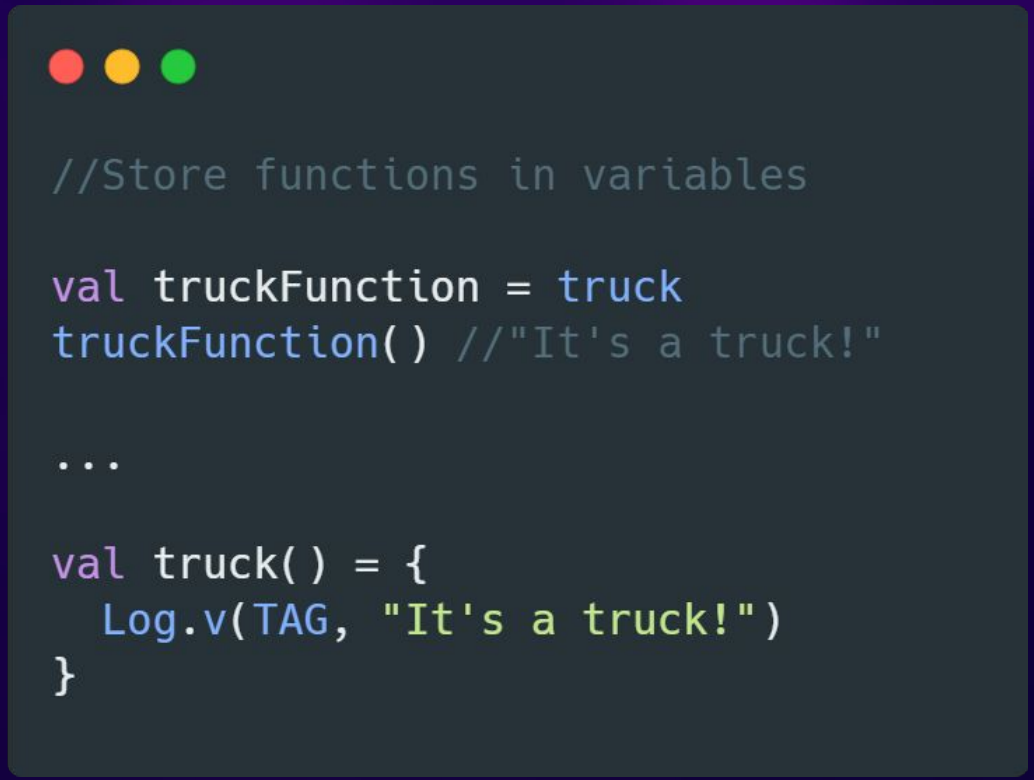
# Store functions in vars/vals

```
//Store functions in variables

val truckFunction = itsAtTruck //ERROR

...

fun itsATruck() {
  Log.v(TAG, "It's a truck!")
}
```

# Store functions in vars/vals



# Store functions in vars/vals



# Store functions in vars/vals

```
//Store functions in variables

val carFunction = carOrTruck(true)
val truckFunction = carOrTruck(false)
carFunction() //"It's a car!"
truckFunction() //"It's a truck!"

...

val truck() = {
  Log.v(TAG, "It's a truck!")
}

val car() = {
  Log.v(TAG, "It's a car!")
}

fun carOrTruck(isCar: Boolean): () -> Unit {
  if (isCar) {
    return car
  }
  else {
    return truck
  }
}
```

unit = void()

# Store functions in vars/vals

```
//Store functions in variables

val vehicleDoors: (Int) -> String = {
  "This vehicle has $it doors"
}
val carFunction = carOrTruck(true, vehicleDoors)
val truckFunction = carOrTruck(false, vehicleDoors)
carFunction() //"It's a car!"
truckFunction() //"It's a truck!"

...

val truck() = {
  Log.v(TAG, "It's a truck!")
}

val car() = {
  Log.v(TAG, "It's a car!")
}

fun carOrTruck(isCar: Boolean, vehicleDoors: (Int) -> String): () -> Unit {
  if (isCar) {
    Log.v(TAG, vehicleDoors(4))
    return car
  }
  else {
    return truck
  }
}
```

# Store functions in vars/vals

```
//Store functions in variables

val vehicleDoors: (Int) -> String = {
  "This vehicle has $it doors"
}
val carFunction = carOrTruck(true, vehicleDoors)
val truckFunction = carOrTruck(false, vehicleDoors)
carFunction() //"It's a car!"
truckFunction() //"It's a truck!"

...

val truck() = {
  Log.v(TAG, "It's a truck!")
}

val car() = {
  Log.v(TAG, "It's a car!")
}

fun carOrTruck(isCar: Boolean, vehicleDoors: ((Int) -> String)?): () -> Unit {
  if (isCar) {
    if (vehicleDoors != null) {
      Log.v(TAG, vehicleDoors(4))
    }
    return car
  }
  else {
    return truck
  }
}
```

es nuleable

# Store functions in vars/vals

```
//Store functions in variables

val vehicleDoors: (Int) -> String = {
  "This vehicle has $it doors"
}
val carFunction = carOrTruck(true, vehicleDoors)
val truckFunction = carOrTruck(false, null)
carFunction() //"It's a car!"
truckFunction() //"It's a truck!"

...

val truck() = {
  Log.v(TAG, "It's a truck!")
}

val car() = {
  Log.v(TAG, "It's a car!")
}

fun carOrTruck(isCar: Boolean, vehicleDoors: ((Int) -> String)?): () -> Unit {
  if (isCar) {
    Log.v(TAG, vehicleDoors?.invoke(4))
    return car
  }
  else {
    return truck
  }
}
```

# Store functions in vars/vals

```
//Store functions in variables

val vehicleDoors: (Int) -> String = {
  "This vehicle has $it doors"
}
val carFunction = carOrTruck(true, vehicleDoors)
val truckFunction = carOrTruck(false)
carFunction() //"It's a car!"
truckFunction() //"It's a truck!"

...

val truck() = {
  Log.v(TAG, "It's a truck!")
}

val car() = {
  Log.v(TAG, "It's a car!")
}

fun carOrTruck(isCar: Boolean, vehicleDoors: ((Int) -> String)? = null): () -> Unit {
  if (isCar) {
    Log.v(TAG, vehicleDoors?.invoke(4))
    return car
  }
  else {
    return truck
  }
}
```

# Lambda functions

```
// Lambda functions

fun MutableList<Int>.findNumber4(onNumberFound: () -> Unit) {
  forEach {
    if (it == 4) {
      onNumberFound()
    }
  }
}

...

(1..10).toMutableList().findNumber4(
  onNumberFound = {
    Log.v(TAG, "Number 4 found!")
  }
)
```