# Kotlin Advanced & Constraint Layout

# Quiz Solutions

*Write a function called countOccurrences that takes a list of strings and returns a map where the key is the string and the value is how many times that string appears in the list.*

```
fun countOccurrences(strings: List<String>): Map<String, Int> {
    val occurrences = mutableMapOf<String, Int>()
    for (str in strings) {
        if (occurrences.containsKey(str)) {
            occurrences[str] = occurrences[str]!! + 1
        } else {
            occurrences[str] = 1
        }
    }
    return occurrences
}
```

*Write a function called calculateSum that takes a list of integers and returns the sum of all even numbers in the list.*

```
fun calculateSum(numbers: List<Int>): Int {
    var sum = 0
    for (num in numbers) {
        if (num % 2 == 0) {
            sum += num
        }
    }
    return sum
}
```

```
fun countOccurrences(strings: List<String>): Map<String, Int> {
 return strings.groupingBy { it }.eachCount()
}
```

**groupingBy** is a transformative operation that works like sorting items into labeled boxes:

- The { it } part is called a lambda expression that tells Kotlin what to group by
- it refers to each individual string in the list
- For each string, it creates a "box" (group) with that string as the label

For more info visit : https://kotlinlang.org/api/core/kotlin-stdlib/kotlin.collections/-grouping/ & https://kotlinlang.org/docs/collection-grouping.html

```
fun calculateSum(numbers: List<Int>): Int {

    return numbers.filter { it % 2 == 0 }.sum()

}
```

The filter function iterates over each element in the collection and applies the provided predicate. If the predicate returns true for an element, that element is included in the resulting collection; otherwise, it is excluded.

For more info : https://kotlinlang.org/docs/collection-filtering.html#partition

# What Are Higher-Order Functions?

A higher-order function is a function that:

1. Takes another function as a parameter
2. Returns a function as a result

✅ **Improves** code reusability

✅ **Encourages** functional programming

✅ **Makes code** more concise & readable

# Common Higher order functions

| Function | Purpose |
| --- | --- |
| `filter` | Selects elements based on a condition |
| `map` | Transforms each element |
| `reduce` | Aggregates elements into a single result |
| `fold` | Similar to `reduce` but allows an initial value |
| `groupBy` | Groups elements based on a condition |

# Filter Example

```
val numbers = listOf(1, 2, 3, 4, 5, 6)

val evenNumbers = numbers.filter { it % 2 == 0 }

println(evenNumbers) // Output: [2, 4, 6]
```

# Map Example

```
val numbers = listOf(1, 2, 3, 4, 5)

val squared = numbers.map { it * it }

println(squared) // Output: [1, 4, 9, 16, 25]
```

# reduce Example

```
val numbers = listOf(1, 2, 3, 4, 5)

val sum = numbers.reduce { acc, num -> acc + num }

println(sum) // Output: 15
```

# fold Example

```
val numbers = listOf(1, 2, 3, 4, 5)

val sumWithInitial = numbers.fold(10) { acc, num -> acc + num }

println(sumWithInitial) // Output: 25
```

# Custom Higher-Order Function Example

Custom function that accepts a function as a parameter.

```
fun applyOperation(a: Int, b: Int, operation: (Int, Int) -> Int): Int {
    return operation(a, b)
}

val multiply = applyOperation(4, 5) { x, y -> x * y }

println(multiply) // Output: 20
```

For more information visit : https://kotlinlang.org/docs/lambdas.html

# Constraint Layout advanced

## Guidelines & Barriers

📌 **Guidelines**: Invisible lines to align multiple views

📌 **Barriers**: Adjust based on **largest** or **smallest** sibling

```
<Guideline
    android:id="@+id/guideline"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    app:layout_constraintGuide_percent=“0.5"/>
```

**Positions elements at 50% of the screen width**

# Constraint Layout - Group

- A virtual view that controls the visibility/behavior of multiple views simultaneously
- Does not provide layout constraints by itself
- Helps manage collections of views efficiently
- Part of the AndroidX ConstraintLayout library

```
<androidx.constraintlayout.widget.ConstraintLayout>
    <androidx.constraintlayout.widget.Group
        android:id="@+id/form_group"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:constraint_referenced_ids="nameInput,emailInput,submitButton" />

    <EditText
        android:id="@+id/nameInput"
        ... />

    <EditText
        android:id="@+id/emailInput"
        ... />

    <Button
        android:id="@+id/submitButton"
        ... />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Chains allow views to be linked together horizontally or vertically

| Chain Mode | Behavior |
| --- | --- |
| **Spread** | Views evenly distributed |
| **Spread Inside** | First & last view fixed, others evenly spaced |
| **Packed** | Views grouped together tightly |

For more information visit https://developer.android.com/develop/ui/views/layout/constraint-layout

# Thank you