Operations quiz (2)

The class Hero

```
data class Hero(
        val name: String,
        val age: Int,
        val gender: Gender?
)
enum class Gender { MALE, FEMALE }
```

#9. Find the result

```
val heroes = listOf(
        Hero("The Captain", 60, MALE),
        Hero("Frenchy", 42, MALE),
        Hero("The Kid", 9, null),
        Hero("Lady Lauren", 29, FEMALE),
        Hero("First Mate", 29, MALE),
        Hero("Sir Stephen", 37, MALE))

val mapByAge: Map<Int, List<Hero>> =
        heroes.groupBy { it.age }
val (age, group) = mapByAge.maxBy { (_, group) ->
    group.size
}!!
println(age)
```

Grouping by a given key

```
val mapByAge: Map<Int, List<Hero>> =
        heroes.groupBy { it.age }

    mapOf(60 to listOf(Hero("The Captain", 60, MALE)),
            42 to listOf(Hero("Frenchy", 42, MALE)),
            9 to listOf(Hero("The Kid", 9, null)),
            29 to listOf(Hero("Lady Lauren", 29, FEMALE),
                Hero("First Mate", 29, MALE)),
            37 to listOf(Hero("Sir Stephen", 37, MALE)))

val (age, group) = mapByAge.maxBy { (_, group) ->
    group.size
}!!
println(age)
```

29

#10. Find the result

```
val heroes = listOf(
        Hero("The Captain", 60, MALE),
        Hero("Frenchy", 42, MALE),
        Hero("The Kid", 9, null),
        Hero("Lady Lauren", 29, FEMALE),
        Hero("First Mate", 29, MALE),
        Hero("Sir Stephen", 37, MALE))


val mapByName: Map<String, Hero> =
        heroes.associateBy { it.name }

mapByName["Frenchy"]?.age
```

Accessing map contents

```
val heroes = listOf(
        Hero("The Captain", 60, MALE),
        Hero("Frenchy", 42, MALE),
        Hero("The Kid", 9, null),
        Hero("Lady Lauren", 29, FEMALE),
        Hero("First Mate", 29, MALE),
        Hero("Sir Stephen", 37, MALE))
```

```
val mapByName: Map<String, Hero> =
        heroes.associateBy { it.name }

mapByName["Frenchy"]?.age
```

42

Accessing map contents

```
val heroes = listOf(
        Hero("The Captain", 60, MALE),
        Hero("Frenchy", 42, MALE),
        Hero("The Kid", 9, null),
        Hero("Lady Lauren", 29, FEMALE),
        Hero("First Mate", 29, MALE),
        Hero("Sir Stephen", 37, MALE))
```

```
val mapByName: Map<String, Hero> =
        heroes.associateBy { it.name }

mapByName["Frenchy"]?.age              // 42

mapByName.getValue("Frenchy").age      // 42
```

map[key] vs map.getValue(key)

```
val mapByName = heroes.associateBy { it.name }


mapByName["unknown"]?.age           // null
mapByName.getValue("unknown").age // NoSuchElementException
```

#11. Find the result

```
val heroes = listOf(
        Hero("The Captain", 60, MALE),
        Hero("Frenchy", 42, MALE),
        Hero("The Kid", 9, null),
        Hero("Lady Lauren", 29, FEMALE),
        Hero("First Mate", 29, MALE),
        Hero("Sir Stephen", 37, MALE))
```

```
val mapByName = heroes.associateBy { it.name }
val unknownHero = Hero("Unknown", 0, null)
mapByName.getOrElse("unknown") { unknownHero }.age
```

associateBy

```
val heroes = listOf(
        Hero("The Captain", 60, MALE),
        Hero("Frenchy", 42, MALE),
        Hero("The Kid", 9, null),
        Hero("Lady Lauren", 29, FEMALE),
        Hero("First Mate", 29, MALE),
        Hero("Sir Stephen", 37, MALE))
```

```
val mapByName = heroes.associateBy { it.name }
val unknownHero = Hero("Unknown", 0, null)
mapByName.getOrElse("unknown") { unknownHero }.age
```

0

#12. Find the result

```
val heroes = listOf(
        Hero("The Captain", 60, MALE),
        Hero("Frenchy", 42, MALE),
        Hero("The Kid", 9, null),
        Hero("Lady Lauren", 29, FEMALE),
        Hero("First Mate", 29, MALE),
        Hero("Sir Stephen", 37, MALE))
```

```
val (first, second) = heroes
        .flatMap { heroes.map { hero -> it to hero } }
        .maxBy { it.first.age - it.second.age }!!
first.name
```

Simplifying code

```
val (first, second) = heroes
        .flatMap { heroes.map { hero -> it to hero } }
        .maxBy { it.first.age - it.second.age }!!
first.name
```

Simplifying code

```
val (first, second) = heroes
        .flatMap { heroes.map { hero -> it to hero } }
        .maxBy { it.first.age - it.second.age }!!
first.name
```

Replace 'it' with explicit parameter

Simplifying code

```
val (first, second) = heroes
        .flatMap { first ->
            heroes.map { hero -> first to hero }
        }
        .maxBy { it.first.age - it.second.age }!!
first.name
```

Simplifying code

```
val (first, second) = heroes
        .flatMap { first ->
            heroes.map { hero -> first to hero }
        }
        .maxBy { it.first.age - it.second.age }!!
first.name
```

Simplifying code

```
val (first, second) = heroes
        .flatMap { first ->
            heroes.map { second -> first to second }
        }
        .maxBy { it.first.age - it.second.age }!!
first.name
```

Simplifying code

Extract variable

```
val (first, second) = heroes
        .flatMap { first ->
            heroes.map { second -> first to second }
        }
        .maxBy { it.first.age - it.second.age }!!
first.name
```

Simplifying code

```
val allPossiblePairs = heroes
        .flatMap { first ->
            heroes.map { second -> first to second }
        }
val (first, second) = allPossiblePairs
        .maxBy { it.first.age - it.second.age }!!
first.name
```

Simplifying code

```
val allPossiblePairs = heroes
        .flatMap { first ->
            heroes.map { second -> first to second }
        }
val (first, second) = allPossiblePairs
        .maxBy { it.first.age - it.second.age }!!
first.name
```

Simplifying code

```
val allPossiblePairs = heroes
        .flatMap { first ->
            heroes.map { second -> first to second }
        }
val (oldest, youngest) = allPossiblePairs
        .maxBy { it.first.age - it.second.age }!!
oldest.name
```

Simplifying code

```
val (first, second) = heroes
        .flatMap { heroes.map { hero -> it to hero } }
        .maxBy { it.first.age - it.second.age }!!
first.name
```

```
val oldest = heroes.maxBy { it.age }
val youngest = heroes.minBy { it.age }
oldest.name
```

The Captain

Simplifying code

- don’t use `it` if it has different types in neighbouring lines
- prefer explicit parameter names if it might be confusing otherwise
- learn the library and try to reuse the library functions as much as possible