Kotlin idioms

1. https://kotlinlang.org/docs/reference/idioms.html#string-interpolation

```
println("Name $name")
```

2. Prefer read-only `val` over mutable `var`

```
/* bad */
var counter = 0
for(i in 0..20){
  counter += i
}

/* preferred */
val sum = (0..20).sum()
```

3. Unit as a return type

```
fun doSomething(): Unit{
  println("do something")
}
```

4. https://kotlinlang.org/docs/reference/idioms.html#single-expression-functions

```
fun theAnswer() = 42
```

5. https://kotlinlang.org/docs/reference/idioms.html#default-values-for-function-parameters

```
fun foo(a: Int = 0, b: String = "") { ... }
```

6. https://kotlinlang.org/docs/reference/idioms.html#creating-a-singleton

```
object Resource {
    val name = "Name"
}
```

7. https://kotlinlang.org/docs/reference/idioms.html#instance-checks

```
when (x) {
    is Foo -> ...
    is Bar -> ...
    else   -> ...
}
```

8. https://kotlinlang.org/docs/reference/idioms.html#creating-dtos-pojospocos

```
data class Customer(val name: String, val email: String)
```

9. Use the fact that `return` and `throw` have type `Nothing`, https://kotlinlang.org/docs/reference/idioms.html#todo-marking-code-as-incomplete

```
val x = foo() ?: return notFound()

val y = if(conditionSatisfied()){
  1.0
} else {
  error("Condition is not satisfied"}//throws inside, the type is Nothing
}
```

https://kotlinlang.org/docs/reference/idioms.html#return-on-when-statement

```
fun transform(color: String): Int {
    return when (color) {
        "Red" -> 0
        "Green" -> 1
        "Blue" -> 2
        else -> throw IllegalArgumentException("Invalid color param value")
    }
}
```

10. Nullable truth.

```
a?.b == true
//instead of
a?.b ?: false
```

11. var x: String?; x = x ?: "Hello"; (analog for dart ?=)

```
var x: String? = null
x = x ?: "Hello"
```

12. Safe chain and elvis (https://kotlinlang.org/docs/reference/idioms.html#if-not-null-and-else-shorthand)

```
val files = File("Test").listFiles()

println(files?.size ?: "empty")
```

13. https://kotlinlang.org/docs/reference/idioms.html#execute-if-not-null

```
val value = ...

value?.let {
    ... // execute this block if not null
}
```

14. https://kotlinlang.org/docs/reference/idioms.html#extension-functions

```
fun String.spaceToCamelCase() { ... }

"Convert this to camelcase".spaceToCamelCase()
```

15. Extension properties

```
import java.util.Calendar

var Calendar.year
    get() = get(Calendar.YEAR)
    set(value) = set(Calendar.YEAR, value)

val calendar = Calendar.getInstance()
println(calendar.year) // 2020
calendar.year = 2021
println(calendar.year) // 2021
```

16. Using with/let/run to create a scope
17. Using apply/also to add a finalizing action
    https://kotlinlang.org/docs/reference/idioms.html#configuring-properties-of-an-object-apply
18. https://kotlinlang.org/docs/reference/idioms.html#read-only-list

```
val list = listOf("a", "b", "c")
```

19. Simple empty collection

```
fun doSomething(additionalArguments: List<String> = emptyList())
```

20. https://kotlinlang.org/docs/reference/idioms.html#accessing-a-map

```
println(map["key"])
map["key"] = value
```

21. Destructuring declaration (https://kotlinlang.org/docs/reference/idioms.html#traversing-a-maplist-of-pairs)

```
for ((k, v) in map) {
    println("$k -> $v")
}

val (a,b) = Pair(1, 2)
```

22. Public var with a private/protected setter

```
var a: Int = 2
        protected set // accessible only in descendents
```

Mutable private - read-only public collections.

```
ptivate val _list = ArrayList<Int>()
val list: List<Int> get() = _list
```

23. https://kotlinlang.org/docs/reference/idioms.html#checking-element-presence-in-a-collection

```
if ("john@example.com" in emailsList) { ... }

if ("jane@example.com" !in emailsList) { ... }
```

24. ranges instead of for loops (https://kotlinlang.org/docs/reference/idioms.html#using-ranges):

```
for (i in 1..100) { ... }  // closed range: includes 100
for (i in 1 until 100) { ... } // half-open range: does not include 100
for (x in 2..10 step 2) { ... }
for (x in 10 downTo 1) { ... }
if (x in 1..10) { ... }
```

25. Trailing lambdas (place lambda the last in the function parameters list, try to provide default values for parameters).Try to make such functions inline (to remove lambda creating overhead).

```
fun integrate(from: Double = 0.0, to: Double = 1.0, function: (Double)Double): Double {...}
integrate{xx*x + 2*x + 1}
integrate(0.0, Pi){ sin(it) }
```

26. .filter.map.reduce (collection processing, higher order functions) (https://kotlinlang.org/docs/reference/idioms.html#filtering-a-list)

```
val list: List<Int> = listOf(1,2,3,4,5,6)

val result = list
        .filter{it%2 == 0} //select even numbers
    .map{it*it} // get square of each element
    .sumByDouble{it.toDouble()} //use one of reduce operations
```

27. Assignment to `if`/`when`/`try`

```
val a = if(b>0) 0 else 1
```

https://kotlinlang.org/docs/reference/idioms.html#trycatch-expression

```
fun test() {
    val result = try {
        count()
    } catch (e: ArithmeticException) {
        throw IllegalStateException(e)
    }

    // Working with result
}
```

```
val ioResult: IOResult? = try{
  readFromFile() //assign result if it read is successful
} catch(t: FileNotFoundException){
  null // return null if file not found
}
```

28. https://kotlinlang.org/docs/reference/idioms.html#builder-style-usage-of-methods-that-return-unit

```
fun arrayOfMinusOnes(size: Int): IntArray {
    return IntArray(size).apply { fill(-1) }
}
```

29. Scoped use of objects (https://kotlinlang.org/docs/reference/idioms.html#java-7s-try-with-resources)

```
val stream = Files.newInputStream(Paths.get("/some/file.txt"))
stream.buffered().reader().use { reader ->
    println(reader.readText())
}
```

30. Companion object. Use companion object as an ID
31. Factory method instead of constructors

```
class Doubles(val list: DoubleArray)

@Suppress("FunctionName")
fun Doubles(vararg numbers: Number) = Doubles(numbers.map{it.toDouble()}.toDoubleArray())

Doubles(1,2,3)
```

32. Lazy properties https://kotlinlang.org/docs/reference/idioms.html#lazy-property

```
val p: String by lazy {
    // compute the string
}
```

33. https://kotlinlang.org/docs/reference/idioms.html#convenient-form-for-a-generic-function-that-requires-the-generic-type-information

```
//  public final class Gson {
//     ...
//     public <T> T fromJson(JsonElement json, Class<T> classOfT) throws JsonSyntaxException {
//     ...

inline fun <reified T: Any> Gson.fromJson(json: JsonElement): T = this.fromJson(json, T::class.java)
```

Non-local return

```
fun foo() {
    listOf(1, 2, 3, 4, 5).forEach {
        if (it == 3) return // non-local return directly to the caller of foo()
        print(it)
    }
    println("this point is unreachable")
}
```

34. DoubleArray vs Array<Double> vs List<Double>

Unsorted

- Read line

```
val map: MutableMap<String, Int>
val (a, b) = readlLine()?.split(" ") ?: error("No line to read") // read line and split it by given
delimeter and use destructuring declaration
map[a] = b.toInt() // Convert the value on-flight and assign it
```

- Use JMH for benchmarking