参考

https://kotlinlang.org/docs/reference/
https://dogwood008.github.io/kotlin-web-site-ja/docs/reference/
https://kotlinlang.org/docs/reference/scope-functions.html#function-selection
https://speakerdeck.com/sys1yagi/jie-pou-kotlin-baitokodowodu-mijie-ku
https://qiita.com/KenjiOtsuka/items/5caaaf2b4344ee8d617b
https://anopara.net/2017/09/16/kotlin%E3%81%AE%E6%B0%97%E3%81%AB%E5%85%A5%E3%82%89%E3%81%AA%E3%81%84%E3%81%A8%E3%81%93%E3%82%8D/
https://kmizu.hatenablog.com/entry/2016/03/22/134828
https://qiita.com/farman0629/items/e8b9bd6fc22d5fa9cfea
https://qiita.com/KenjiOtsuka/items/5caaaf2b4344ee8d617b
https://qiita.com/farman0629/items/e8b9bd6fc22d5fa9cfea
https://blog.y-yuki.net/entry/2019/05/13/100000

概要

JVM で動作する言語。Java の文法の冗長的なところを書きやすくした感じ。
Java と互換性があり、通常の Java のコードを呼び出したりすることもできる。
jad でデコンパイルすると、至って普通の Java のコードになっている。

環境

https://github.com/JetBrains/kotlin/releases/tag/v1.3.72
からダウンロードしてパスを通す。

sdkman を使っても良い
https://sdkman.io/

その他

困ったら

基本的には Java なので、jad でデコンパイルすると実際は何をしているのかわかりやすい。

ラムダ式と無名関数

似ているけど、記述と return の動作が違う。

■記述

ラムダ式

```
var f: (Int, Int) -> Int = { a: Int, b:Int -> a + b }
println(f(1,2))
// return は書けない。
```

無名関数

```
var f2: (Int, Int) -> Int = fun(a:Int, b:Int):Int {
  return a + b
}
// return は必要 ( 戻り値がある場合 )
```

■ return の動作

ラムダ式

```
var l = listOf(1,2,3)
l.forEach {
  if (it == 2) return // ここで呼び出し元を return してしまう
  println(it)
}
```

無名関数

```
l.forEach(fun(it) {
  if (it == 2) return //continue と同じ。次の値に処理が移る
  println(it)
})
```

コープ関数

https://kotlinlang.org/docs/reference/scope-functions.html#function-selection
let, run, also, apply は、それぞれ引数と戻り値が違うので状況により使い分けする。
ただし、run は 2 種類 (拡張関数ではないものもある) あるので注意。