Kotlin

・https://kotlinlang.org

・リファレンス

Install

Command Line

■ SDKMAN

・http://sdkman.io/
・https://qiita.com/saba1024/items/967ee3d8a79440a97336

```
$ curl -s https://get.sdkman.io | bash
```

・ターミナルを開き直し

```
$ sdk install kotlin
```

・確認

```
$ kotolinc -help
```

■ REPL

```
$kotlinc-jvm
```

IDE

■ Intellij IDEA

・https://www.jetbrains.com/idea/

基本

コンパイルと実行

・コンパイル

```
$ kotlinc sample.kt
```

・実行

```
$ kotlin sample.SampleKt
```

データ型

■数値型

型
Double
Float
Long
Int
Short
Byte

パッケージと関数

```
// ソースファイルの先頭に記述
// ディレクトリ構造と一致する必要はない。
package sample

// 2 つの Int 引数と戻り値 Int をもつ
fun sum(a: Int, b: Int): Int {
    return a + b
}

// 式を持ち、戻り値を推測
fun sum2(a: Int, b: Int) = a + b

// 意味のある値を返さない場合、Unit を使用
fun printSum(a: Int, b: Int): Unit {
    println("$a + $b = ${a + b}")
}

// Unit は省略可能
fun printSum2(a: Int, b: Int) {
    println("$a + $b = ${a + b}")
}

fun main(args: Array<String>) {
    println(sum(1,2))
    println(sum2(3,4))
    printSum(5,6)
    printSum(7,8)
}
```

実行

```
PS> kotlinc Test01.kt
PS> kotlin sample.Test01Kt
3
7
5 + 6 = 11
7 + 8 = 15
```

ローカル変数

```
package sample

fun main(args: Array<String>) {
    // val: 読み取り専用
    val a: Int = 1 // 即時割り当て
    val b = 2 // Int 型を推測
    val c: Int // 即時初期化しない場合、型は必須
    c = 3
    // c = 4 // error: val cannot be reassigned

    // var: 変更可能
    var d = 5
    d += 1
```

```
    println("$a,$b,$c,$d")
}
```

文字列テンプレート

```
package sample

fun main(args: Array<String>) {
    // 文字列はテンプレート式を含むことができる
    // テンプレート式は、$ で始める
    val a = 10
    println("a is $a")

    // 中括弧で任意の式を表現
    val s = "abc"
    println("$s.length is ${s.length}")

    // テンプレートは、raw 文字列、エスケープされた文字列もサポートする
    // $ リテラルを表示するのに、バックスラッシュエスケープは機能しない ( 以下参照 )
    val l = """
    ${'$'}123
    """
    print(l)
}
```

実行

```
PS> kotlin sample.Test03Kt
a is 10
abc.length is 3

    $123
```

条件式

・三項演算子はないが、if を式として利用できる

```
package sample

fun maxOf(a: Int, b: Int): Int {
    if (a > b) {
        return a
    } else {
        return b
    }
}

// if を式として利用
fun maxOfExpression(a: Int, b: Int) = if (a > b) a else b

fun main(args: Array<String>) {
    println(maxOf(3,5))
    println(maxOfExpression(6,8))
}
```

実行

```
PS> kotlin sample.Test04Kt
5
8
```

nulable と null チェック

ターミナルで、kotlinc とタイプすると、REPL が起動する。終了には、:quit

・参照は、null が設定可能な場合、明示的に nullable とマークする必要がある

```
>>> var a: Int? =null
>>> a
null
```

・マークしないとエラー

```
 >>> var a: Int = null
error: null can not be a value of a non-null type Int
var a: Int = null
```

安全呼び出し

・?. を使うと、オブジェクトが null の場合、null を返し、そうでない場合メソッドを呼び出す

```
var foo:Foo? = bar?.getInstance()
```

エルビス演算子

・?: を使用すると、オブジェクトが null の場合に返す値を指定できる

```
var i:Int ? = null
println("%d".format(i ?: -1))
-1
```

・items(List など) の null チェックを実施し、null なら 0 の例

```
return items?.size ?:0
```

強制 Not null

・!! を使用すると、nullable に対して、強制的に not null

```
var foo:String? = null
var bar:String = foo!!
tlin.KotlinNullPointerException
```

型チェックと自動キャスト

・is 演算子はインスタンスの型をチェックする
・イミュータブルなローカル変数、プロパティは型がチェック済みのため明示的なキャストは不要

```
package sample

fun getStringLen(obj: Any):Int? {
    if (obj is String) {
        // obj は、型チェックブロック内で自動的にキャストされる
        return obj.length;
    }
    // 型チェックブロックの外では、Any のまま
    return null
}
```

```
fun main(args: Array<String>) {
    println(getStringLen("abc"))
    println(getStringLen(123))
}
```

実行

```
PS> kotlin sample.Test05Kt
3
null
```

for ループ

```
package sample

fun main(args: Array<String>) {
    var items = listOf("a", "b", "c")
    for (item in items) {
        println(item)
    }

    for(index in items.indices) {
        println("item[$index] is ${items[index]}")
    }
}
```

実行

```
> kotlin sample.Test06Kt
a
b
c
item[0] is a
item[1] is b
item[2] is c
```

```
for (year in 1900..2018 step 1) {
    println(convertJyunishiYear(year))
}
```

while ループ

```
var index = 0
while(index < items.size) {
    println("item[$index] is ${items[index]}")
    index++
}
```

When 式

・Kotlin には switch はない

引数あり

```
package sample

fun describe(obj: Any?): String =
```

```
    when(obj) {
        1           -> "One"
        "Hello"     -> "Greeting"
        is Long     -> "Long"
        !is String  -> "Not a string"
        else        -> "Unknown"
    }

    fun main(args: Array<String>) {
        val items = listOf("Hello",1,9999999999,10.1,"hoge")
        for(item in items) {
            println("$item is ${describe(item)}")
        }
    }
```

実行

```
    PS> kotlin sample.Test07Kt
    Hello is Greeting
    1 is One
    9999999999 is Long
    10.1 is Not a string
    hoge is Unknown
```

引数なし

・https://qiita.com/AAkira/items/3d5b694d488fe029d7b9

```
    when {
        hoge == 0 -> {
            println("0")
        }
        hoge == 1 || hoge == 2 -> {
            println("1, 2")
        }
        else -> {
            println("else")
        }
    }
```

Rage

```
    package sample

    fun main(args: Array<String>) {
        for (x in 1..3) {
            println("iteration $x")
        }

        for (x in 1..6 step 2) {
            println("step $x")
        }

        for (x in 6 downTo 1 step 2) {
            println("downTo $x")
        }

        // 値が Range にの範囲内か in を使用してチェック
        val x = 10
        val y = 9
        if (x in 1..y+1) {
            println("$x in range")
        }

        val index = -1
        val list = listOf("a","b","c")
        if (index !in 0..list.lastIndex) {
            println("$index is out of range")
        }
```

```
}
```

実行

```
PS> kotlin sample.Test08Kt
iteration 1
iteration 2
iteration 3
step 1
step 3
step 5
downTo 6
downTo 4
downTo 2
10 in range
-1 is out of range
```

Try-with-resouce

```
val stream = Files.newDirectoryStream(FileSystems.getDefault().getPath(targetDir))
stream.use {
    for (path in stream) {
        println(path.fileName)
    }
}
```

クラスとオブジェクト

プロパティ

```
class Address {
    var name: String = ...
    var street: String = ...
    var city: String = ...
    var state: String? = ...
    var zip: String = ...
}
```

列挙

```
enum class Jikkan {
    Kinoe,
    Kinoto,
    Hinoe,
    Hinoto,
    Tuchinoe,
    Tuchinoto,
    Kanoe,
    Kanoto,
    Mizunoe,
    Mizunoto
}
```

・初期化

```
enum class Jikkan(val value: String) {
    Kinoe("甲"),
    Kinoto("乙"),
    Hinoe("丙"),
    Hinoto("丁"),
    Tuchinoe("戊"),
    Tuchinoto("己"),
    Kanoe("庚"),
    Kanoto("辛"),
    Mizunoe("壬"),
    Mizunoto("癸")
```

```
}
```

書式

```
"%s%s".format(foo,bar)
```

コレクション

・Kotlin のコレクション使い方メモから引用

■書き込み可能なコレクション

```
val list = mutableListOf(1, 2, 3)
val map = mutableMapOf("foo" to "FOO", "bar" to "BAR")
val set = mutableSetOf(9, 8, 7)
```

■読み取り専用コレクション

```
val list = listOf(1, 2, 3)
val map = mapOf("foo" to "FOO", "bar" to "BAR")
val set = setOf(9, 8, 7)
```

■ Java のコレクションを生成

```
var ia = arrayListOf<Int>()
var hs = hashSetOf<Int>()
var hm = hashMapOf<Int, Int>()
```

イディオム

DTO(POJO/POCO)

・以下が提供される
  ・プロパティに対する getter/setter
  ・equals(),hashCode(),toString(),copy()
  ・component1(),component2(),...componentN() 全プロパティ

```
package sample

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

fun main(args: Array<String>) {
    val cust = Customer("hoge","hoge@hoge.com")
    println(cust.toString())
    println("component1=${cust.component1()}")
    println("component2=${cust.component2()}")
    val(name,email) = cust
    print("name=$name,email=$email")
}
```

実行

```
PS> kotlin sample.Idiom01Kt
Customer(name=hoge, email=hoge@hoge.com)
component1=hoge
component2=hoge@hoge.com
name=hoge,email=hoge@hoge.com
```

関数パラメータデフォルト値

```
fun foo(a: Int=1, b: String="default") {
    println("a=$a,b=$b")
}

fun main(arg: Array<String>) {
    foo()
    foo(9,"custom")
}
```

リストのフィルタ

```
package sample

fun main(args: Array<String>) {
    // 変更可能なリスト
    val mutableList: MutableList<Int> = mutableListOf()
    for(n in 1..10) {
        mutableList.add(n)
    }
    println(mutableList)
    // リストのフィルタ
    println(mutableList.filter{x -> x % 2 == 0})
}
```

実行

```
PS> kotlin sample.Idiom03Kt
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[2, 4, 6, 8, 10]
```

Kotlin Android

・Kotlin Android

Tips

File

■読み込む

```
val reader = File(path)
reader.forEachLine {
    println(it)
}
```

■指定のフォルダのファイル全てを読む

```
import java.io.File
import java.nio.file.*

class EachReader {

    fun check(targetDir: String) {

        val stream = Files.newDirectoryStream(FileSystems.getDefault().getPath(targetDir))
        stream.use {
            for (path in stream) {

                val fd = path.toFile()
                fd.forEachLine {
```

```
                    println(it)
                }
            }
        }
    }
}

fun main(args: Array<String>) {
    val targetDir = """C:\work"""
    var checker = EachReader()
    checker.check(targetDir)
}
```

■固定バイトで読み込む

```
val reader = File(path).reader()
var buf:CharArray = CharArray(140)
while(reader.read(buf) > 0) {
    val line = String(buf)
    println(line)
}
```

■文字コードを指定して読み書き

読み

```
val reader = File(file).bufferedReader(Charset.forName("Shift_JIS"))
```

書き

```
val writer = File("out.csv").bufferedWriter(Charset.forName("Shift_JIS"))
```

Sample

・Android + Kotlin で Firebase によるメール認証

Link

・kotlin 標準ライブラリ