Swift **&/vs** Kotlin – Syntax Comparison

Hello World

Swift

print("Hello, world!")

Kotlin

println("Hello, world!")

Variables & Constants

Swift

var myVar = 42

myVar = 50

let myConstant = 42

Kotlin

var myVar = 42

myVar = 50

val myConstant = 42

Explicit Types

Swift

let explicitDouble: Double = 70

Kotlin

val explicitDouble: Double = 70

Type Coercion

Swift

let label = "The width is "

let width = 94

let widthLabel = label + String(width)

Kotlin

val label = "The width is "

val width = 94

val widthLabel = label + width

String Interpolation

Swift

let apples = 3

let oranges = 5

let fruitSummary = "I have \(apples + oranges) " + "pieces of fruit."

Kotlin

val apples = 3

val oranges = 5

val fruitSummary = "I have ${apples + oranges} " +"pieces of fruit."

Range Operator

Swift

let names = ["Mark", "Anshul", "Dmitri", "Luke"]

let count = names.count

for i in 0..<count { print("Person \(i + 1) is called \(names[i])") }

// Person 1 is called Mark // Person 2 is called Anshul // Person 3 is called Dmitri // Person 4 is called Luke

Kotlin

val names = arrayOf("Mark", "Anshul", "Dmitri", "Luke")

val count = names.count()

for (i in 0..count - 1) { println("Person ${i + 1} is called ${names[i]}") }

// Person 1 is called Mark // Person 2 is called Anshul // Person 3 is called Dmitri // Person 4 is called Luke

Inclusive Range Operator

Swift

for index in 1...5 { print("\(index) times 5 is \(index \* 5)") }

// 1 times 5 is 5 // 2 times 5 is 10 // 3 times 5 is 15 // 4 times 5 is 20 // 5 times 5 is 25

Kotlin

for (index in 1..5) { println("$index times 5 is ${index \* 5}") }

// 1 times 5 is 5 // 2 times 5 is 10 // 3 times 5 is 15 // 4 times 5 is 20 // 5 times 5 is 25

COLLECTIONS

Arrays

Swift

var shoppingList = ["fruits", "water", "grocery", "disposables"]

shoppingList[1] = "glass of water"

Kotlin

val shoppingList = arrayOf("fruits", "water", "grocery", "disposables")

shoppingList[1] = "glass of water"

Maps

Swift

var designations = [ "Malcolm": "Captain", "Kaylee": "Mechanic" ]

designations ["Jayne"] = "Public Relations"

Kotlin

val designations = mutableMapOf( "Malcolm" to "Captain", "Kaylee" to "Mechanic" )

designations ["Jayne"] = "Public Relations"

Empty Collections

Swift

let emptyArray = [String]()

let emptyDictionary = [String: Float]()

Kotlin

val emptyArray = arrayOf<String>()

val emptyMap = mapOf<String, Float>()

FUNCTIONS

Functions

Swift

func greet(\_ name: String,\_ day: String) -> String { return "Hello \(name), today is \(day)." }

greet("Bob", "Tuesday")

Kotlin

fun greet(name: String, day: String): String { return "Hello $name, today is $day." }

greet("Bob", "Tuesday")

Tuple Return

Swift

func getGasPrices() -> (Double, Double, Double) { return (3.59, 3.69, 3.79) }

Kotlin

data class GasPrices(val a: Double, val b: Double, val c: Double)

fun getGasPrices() = GasPrices(3.59, 3.69, 3.79)

Variable Number Of Arguments

Swift

func sumOf(\_ numbers: Int...) -> Int {

var sum = 0

for number in numbers {

sum += number

}

return sum

}

sumOf(42, 597, 12)

Kotlin

fun sumOf(vararg numbers: Int): Int {

var sum = 0

for (number in numbers) { sum += number }

return sum }

sumOf(42, 597, 12) //

sumOf() can also be written in a shorter way: fun sumOf(vararg numbers: Int) = numbers.sum()

Function Type

Swift

func makeIncrementer() -> (Int -> Int) {

func addOne(number: Int) -> Int { return 1 + number }

return addOne }

let increment = makeIncrementer() increment(7)

Kotlin

fun makeIncrementer(): (Int) -> Int {

val addOne = fun(number: Int): Int { return 1 + number } return addOne }

val increment = makeIncrementer() increment(7)

Map (Higher Order Functions)

Swift

let numbers = [20, 19, 7, 12]

numbers.map { 3 \* $0 }

Kotlin

val numbers = listOf(20, 19, 7, 12)

numbers.map { 3 \* it }

Sort

Swift

var mutableArray = [1, 5, 3, 12, 2]

mutableArray.sort()

Kotlin

listOf(1, 5, 3, 12, 2).sorted()

Named Arguments

Swift

func area(width: Int, height: Int) -> Int { return width \* height }

area(width: 2, height: 3)

Kotlin

fun area(width: Int, height: Int) = width \* height area(width = 2, height = 3) // This is also possible with named arguments area(2, height = 2) area(height = 3, width = 2)

CLASSES

Declaration

Swift

class Shape { var numberOfSides = 0

func simpleDescription() -> String { return "A shape with \(numberOfSides) sides."

} }

Kotlin

class Shape { var numberOfSides = 0

fun simpleDescription() = "A shape with $numberOfSides sides." }

Usage

Swift

var shape = Shape()

shape.numberOfSides = 7

var shapeDescription = shape.simpleDescription()

Kotlin

var shape = Shape()

shape.numberOfSides = 7

var shapeDescription = shape.simpleDescription()

Subclass

Swift

class NamedShape { var numberOfSides: Int = 0

let name: String

init(name: String) { self.name = name }

func simpleDescription() -> String { return "A shape with \(numberOfSides) sides." } }

class Square: NamedShape {

var sideLength: Double

init(sideLength: Double, name: String) {

self.sideLength = sideLength

super.init(name: name)

self.numberOfSides = 4 }

func area() -> Double { return sideLength \* sideLength }

override func simpleDescription() -> String { return "A square with sides of length " + sideLength + "." } }

let test = Square(sideLength: 5.2, name: "square")

test.area()

test.simpleDescription()

Kotlin

open class NamedShape(val name: String) {

var numberOfSides = 0

open fun simpleDescription() = "A shape with $numberOfSides sides." }

class Square(var sideLength: BigDecimal, name: String) : NamedShape(name) {

init { numberOfSides = 4 }

fun area() = sideLength.pow(2)

override fun simpleDescription() = "A square with sides of length $sideLength." }

val test = Square(BigDecimal("5.2"), "square")

test.area()

test.simpleDescription()

Checking Type

Swift

var movieCount = 0

var songCount = 0 for item in library { if item is Movie { movieCount += 1 } else if item is Song { songCount += 1 } }

Kotlin

var movieCount = 0

var songCount = 0 for (item in library) { if (item is Movie) { ++movieCount } else if (item is Song) { ++songCount } }

Pattern Matching

Swift

let nb = 42

switch nb {

case 0...7, 8, 9: print("single digit")

case 10: print("double digits")

case 11...99: print("double digits")

case 100...999: print("triple digits")

default: print("four or more digits")

}

Kotlin

val nb = 42 when (nb)

{ in 0..7, 8, 9 -> println("single digit")

10 -> println("double digits")

in 11..99 -> println("double digits")

in 100..999 -> println("triple digits")

else -> println("four or more digits") }

Downcasting

Swift

for current in someObjects { if let movie = current as? Movie { print("Movie: '\(movie.name)', " + "dir. \(movie.director)") } }

Kotlin

for (current in someObjects) { if (current is Movie) { println("Movie: '${current.name}', " + "dir. ${current.director}") } }

Protocol

Swift

protocol Nameable { func name() -> String } func f<T: Nameable>(x: T) { print("Name is " + x.name()) }

Kotlin

interface Nameable { fun name(): String } fun f<T: Nameable>(x: T) { println("Name is " + x.name()) }

Extensions

Swift

extension Double { var km: Double { return self \* 1\_000.0 }

var m: Double { return self }

var cm: Double { return self / 100.0 }

var mm: Double { return self / 1\_000.0 }

var ft: Double { return self / 3.28084 } }

let oneInch = 25.4.mm print("One inch is \(oneInch) meters") // prints "One inch is 0.0254 meters"

let threeFeet = 3.ft print("Three feet is \(threeFeet) meters") // prints "Three feet is 0.914399970739201 meters"

Kotlin

val Double.km: Double get() = this \* 1000

val Double.m: Double get() = this

val Double.cm: Double get() = this / 100

val Double.mm: Double get() = this / 1000

val Double.ft: Double get() = this / 3.28084

val oneInch = 25.4.mm println("One inch is $oneInch meters") // prints "One inch is 0.0254 meters"

val threeFeet = 3.0.ft println("Three feet is $threeFeet meters") // prints "Three feet is 0.914399970739201 meters"

Reference:

http://angelolloqui.com/blog/38-Swift-vs-Kotlin-for-real-iOS-Android-apps