Kotlin Cheat Sheet

by scottstoll2017 (ScottHOC) via cheatography.com/45425/cs/13463/

General Tips

Types are capitalized	You do NOT need ; at the end of lines.	Java uses "switch", Kotlin uses "when".
Kotlin reads like English. When you see this:	Think this:	Example:
:	"Is type"	var age : Int = 10
->	"returns"	when (x) is 12 -> "It's a dozen"

Packages and Imports are best understood by reading the info here: Packages and Imports

Variables

var is a Variable that can be changed.	val is a Value that never changes, like your name.
val a: Int = 1	a is initialized and it's type is specified.
var b = 2	b is initialized and it's type is ***inferred***. (You do *not* need to specify type when you declare unless the compiler guesses wrong. Normally the compiler is very good at inferring the type.)
var c : Int	c isn't initialized so it's type *must* be specified. **This can lead to nulls which are evil JuJu in Kotlin.** (Bad Dev, no coffee for you.)
var d : String? = "Nullable"	The '?' means this is specified with a **nullable** string. (You are forcing the compiler to allow a null value... why?? Don't do this unless you have a compelling reason.)
var e : String = "notNullable"	This is specified with a non-nullable string.

Avoid nulls if at all possible. This is because in Kotlin a great deal of effort has gone into trying to eliminate null pointer exceptions and the null safety that is one of Kotlin's greatest assets is undermined if you intentionally let things be nullable by using the '?'

Number Types

The usual:	**Type**	**Bit Width**
	Double	64
	Float	32
	Long	64
	Int	32
	Short	16
	Byte	8
Floating Point Notation:		

Number Types (cont)

	Longs are tagged with 'L'	1000L
	Floats are tagged with 'f' or 'F'	1000f 1000F
Literal Constants:	Decimal	100 125.5
	Hex	0xFF342
	Binary	0b101101
	Octal	*Not Supported*
Range	range = 1..10	Contains all Integers from 1 to 10.

Since Kotlin 1.1 you can make numbers more readable with underscores:
val oneMillion = 1_000_000
val creditCardNumber = 1234_5678_9012_3456L
val socialSecurityNumber = 999_99_9999L
val hexBytes = 0xFF_EC_DE_5E
val bytes = 0b11010010_01101001_10010100_10010010

Visibility Modifiers:

Public can be seen by anyone, "who sees the declaring class".

Internal can only be accessed from within the same module / package. (This is great for library authors and should be the standard for Android apps that have all the code in a single package.)

Protected is visible inside the class AND subclasses.

Private is visible inside the class only.

Note: In an Android app that has all of it's code in one package there is never any reason to have anything be **Public**. Instead, make all of your declarations **Internal**, **Protected** or **Private**.

Comments

// Line Comment	/* Block Comment */	/** KDoc Comment */

Null Safety

var a : String = null	You know better by now. This won't fly.
var a : String? = null	The '?' says you want to allow for the possibility of a null. It's not recommended unless you *have* to make an allowance for a null, such as interoperability with Java code that is already written to allow for nulls.

Some things can cause issues, such as checking the length of a declared but uninitialized array and assigning it to a non-nullable Int:	`var a : Array`
	`var b : Int = a.length`
	Is a recipe for a crash.

Null Safety (cont)

Checking for a null first prevents us from shooting ourselves in the foot. We have a few ways to do it.	Using an If:
	`if(a != null) a.length else -1`
	
	Single line if's are easier than the classic if which would read:
	`if(a != null){`
	`     return a.length`
	`     } else {`
	`     return -1`
	`}`

	Safe Calls. "Verify that this isn't a null before doing anything."
	`a?.length`
	In context, this declaration says that lengthOfArray is not allowed to be null but then the code uses a Safe Call (?) to check the length of an array that wasn't initialized:
	`var a : Array`
	`lengthOfArray : Int = a?.length`

Trying to assign a nullable to a non-nullable causes your computer to leak magic smoke.	`var a =5`
	`var b? : Int = 10`
	`a = b`
	//No good. Can't do that*even though neither one is null* because 'a' is non-nullable but 'b' is nullable. This is because 'b' could potentially be null and 'a' cannot, so this won't fly even though a value was assigned.
	*Think of `Int` and `Int?` as two different types and you need to cast an `Int` to an `Int?`, because you do.

And we all know that assigning a non-nullable value to a nullable is fine.	`var a : Int?`
	`var b = 5`
	`a = b`
	// We do this all the time

Flow Control

If, Else If, Else

```
if (a > b) {
     return a
     } else if ( b < c ) {
     return b
     } else {
     magicSmokeLeakedOutOfLaptop()
}
```

When (It's called Switch in Java and C#)

```
when (x) {

     "Hot Dog" -> print("Mustard")

     true -> lie = false

     42 -> "Secret is that none of the selections are in {} code blocks."

     else -> { tip =  "But notice that the else statement
                    IS in a code block"}

}
```

Nested For Loops

```
fun forRange() {

     // Run 3 Outer Loops
     for (item in 1..3) {
          println("Outer loop #$item \n\nThe inner loop is:")

          for (item in 1..10) {
               // in the inner loop, print the even numbers in the range 1..10
               if (item % 2 == 0) {
               println("$item is even.")
          }
          }
          print("Inner loop finished.\n\n")
     }
}
```

For Each

```
fun forArray() {
     var testArray: Array<String> = arrayOf("First", "Second", "Third", "Fourth", "Fifth")
     // For each thing in the array, print each one.
     for (each in testArray) {
          // Yes, it's this simple
          println(each)
     }
}
```

Sample While

```
fun sampleWhile() {
     var count = 1

     println("\nSample While:")

     while (count <= 5) {
          println("Count is: $count")

          count++
     }

}
```

Sample Out of Range While

```
// A while loop might never execute if it's test fails
fun sampleOutOfRangeWhile() {
     var countIsTooLarge = 10

     println("\nSample Out of Range While (Won't Print)")

     // This will never execute at all because the condition
     // is tested before the code is executed.
     while (countIsTooLarge <= 5)
          println("Count too large count is: $countIsTooLarge")
     countIsTooLarge++

}
```

Sample Do While

```
fun sampleDoWhile() {
     var count = 1

     println("\nSample Do While")

     do {
          println("Do While count is: $count")
          count++
     } while (count <= 5)
}
```

Sample Out of Range Do While

```
// A Do While will always exectue at least once because
// it's test isn't performed until after the block executes.
```

```
fun sampleOutOfRangeDoWhile() {
     var count = 1

     println("\nSample Out of Range Do While\n(Prints once before condition is checked)")

     do {
          println("Do While count is: $count")
          count++
     } while (count >= 5)
}
```

Continue and Break

//Continue:

```
fun main(args: Array<String>) {

     println("Using a not equal test to skip #4:\n")
     useNotEqualToSkip()
     println("\nUsing continue to skip when count = 4:\n")
     useContinueToSkip()
     println("\nYou should see the exact same output.")

     println("\nNow let's use break to skip out after #3:\n")
     useBreakToGetOut()

}

fun useNotEqualToSkip() {
     for (count in 1..5) {
          if (count != 4) {
               println(count)
     }
     }
}

fun useContinueToSkip() {
     for (count in 1..5) {
          if (count == 4) {
               continue
          }
          println(count)
     }
}
```

//Break

// This code will break you out of the loop after #3

```
fun useBreakToGetOut() {
     for (count in 1..5) {
          if (count == 4) {
               break
          }
          println(count)
     }
}
```

In short:

continue: Go back to the beginning of the block (the for loop)
break: Break out of the block completely and move on. (Leave the loop.)

For continue there are two examples, one uses a != test to skip over 4 and the other uses an if to check if the count is = 4 and then utilizes a continue to skip the rest of the code block and just go back to the beginning of the loop.

The break is simple enough. When the break is called you exit the for loop entirely.

By **scottstoll2017** (ScottHOC)
cheatography.com/scotthoc/