Universidad Nacional de La Matanza

Kotlin

Introducción y primeros pasos

Agenda

- Sobre el lenguaje
- Funciones
- Variables mutables e inmutables
- Strings
- Condicionales
- Estructuras de repetición
- Clases
- Objetos
- ¿Cómo seguir?

¿Por qué un nuevo lenguaje?

- En Android es difícil soportar nuevas versiones de Java
  - Java 7
- Desarrollado por JetBrains… apoyado por Google.
  - ¿Conflicto inminente con Oracle?
- Buena recepción en la comunidad de desarrolladores

El lenguaje

- Moderno y expresivo
- Más seguro
- Interoperable

Arranquemos con lo básico

- Archivos .kt
- Definición de packages, como en Java pero más flexible
  - No es necesario que la carpeta donde está el archivo coincida con el package declarado
- Puede declararse más de una clase por archivo
- Pueden declararse funciones sin que pertenezcan a ninguna clase
  - Las funciones son *first-class citizen*.
- No es necesario que los statements terminen con punto y coma

Operadores

Operadores

- Operadores matemáticos `+ - * / %`
- Operadores de crecimiento y decrecimiento `++ --`
- Operadores de comparación `< <= > >=`
- Operador de asignación `=`
- Operadores de igualdad `== !=`

Operaciones con enteros

```
1 + 1      =>          2

53 - 3     =>          50

50 / 10    =>          5

9 % 3      =>          0
```

Operaciones con decimales

```
1.0 / 2.0   =>          0.5

2.0 * 3.5   =>          7.0
```

Operadores matemáticos

```
1+1
⇒ kotlin.Int = 2
```

```
53-3
⇒ kotlin.Int = 50
```

```
50/10
⇒ kotlin.Int = 5
```

```
1.0/2.0
⇒ kotlin.Double = 0.5
```

```
2.0*3.5
⇒ kotlin.Double = 7.0
```

⇒ indicates output from your code.

Result includes the type (`kotlin.Int`).

Numeric operator methods

Kotlin mantiene a los números como primitivos. Pero deja llamar métodos a estos números como si fuesen objetos.

```
2.times(3)
⇒ kotlin.Int = 6

3.5.plus(4)
⇒ kotlin.Double = 7.5

2.4.div(2)
⇒ kotlin.Double =
1.2
```

Data types

Tipos de Enteros

Type	Bits	Notes
Long	64	From -2^{63} to $2^{63}-1$
Int	32	From -2^{31} to $2^{31}-1$
Short	16	From -32768 to 32767
Byte	8	From -128 to 127

Decimales y otros tipos numéricos

Type	Bits	Notes
Double	64	16 - 17 dígitos significantes
Float	32	6 - 7 dígitos significantes
Char	16	16-bit Unicode character
Boolean	8	True or false. Las operaciones incluyen: || - lazy disjunction, && - lazy conjunction, ! - negation

Funciones

Sobre functions

- Son un bloque de código que ejecuta una tarea específica
- Rompe el programa en pedazos modulares más pequeños
- Se declaran usando la palabra clave `fun`
- Pueden recibir argumentos que sean nombrados o con valores por defecto

Partes de una función

Previamente, creaste una función simple que imprimía "Hello World".

```
fun printHello() {
    println("Hello World")
}

printHello()
```

Lo básico

```
fun sum(a: Int, b: Int): Int {
    return a + b
}
```

Si sólo retornan un valor

```
fun sum(a: Int, b: Int) = a + b
```

Si no retornan nada

Variables mutables e inmutables

Inmutables

```
fun ejemplo() {
    val a: Int = 1  // Se inicializa en el momento
    a = 4;
}

    val b = 2   // Se infiere el tipo

    val c: Int  // No puede declararse y no inicializarse
    print(c)

}
```

Val cannot be reassigned

Mutables

```
fun ejVariables() {

    var x = 5
    x += 1
    x++
    x = 20
}
```

Strings

String templates

```
fun ejStrings() {

    val a = 1
    val s1 = "a vale $a"
    val s2 = "a es mayor a 5? ${a > 5}"
}
```

Condicionales

if

```
fun maxOf(a: Int, b: Int): Int {

    if (a > b) {
        return a
    } else {
        return b
    }
}
```

If como expresión

```
fun maxOf(a: Int, b: Int) = if (a > b) a else b
```

when

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

Estructuras de repetición

for

```
fun saludos() {

    val nombres = listOf("Facundo", "Maribel", "Lara")

    for (nombre in nombres) {
        println("Hola, $nombre")
    }

}
```

for

```
fun saludos() {

    val nombres = listOf("Facundo", "Maribel", "Lara")

    for (indice in nombres.indices) {
        println("Hola, ${nombres[indice]}")
    }

}
```

for

```
fun saludos() {

    val nombres = listOf("Facundo", "Maribel", "Lara")

    for (i in 0..2) {
        println("Hola, ${nombres[i]}")
    }

}
```

while

```
fun saludos() {

    val nombres = listOf("Facundo", "Maribel", "Lara")

    var i = 0
    while (i < nombres.size) {
        println("Hola, ${nombres[i]}")
        i++
    }

}
```

Clases

Clase básica y creación de instancia

```
class Persona
val nuevaPersona = Persona()
```

Constructor principal

```
class Persona(val nombre: String)
val nuevaPersona = Persona( nombre: "Facundo")
```

Recibiendo atributos en el constructor

```
class Persona(private val nombre: String) {

    fun saludar() = "Hola, me llamo $nombre"

}

val nuevaPersona = Persona( nombre: "Facundo")
nuevaPersona.saludar()
```

Herencia

- Herencia implícita: Todos los objetos heredan de *Any* (en Java esto es así con *Object*)
  - equals(), hashCode() y toString()

- Para que se pueda heredar de una clase, esta debe ser marcada con la palabra reservada *open* (esto es exactamente al revés que en Java, donde hay que marcar las clases para que **no** se pueda heredar de ellas con *final*)

Herencia

```
open class Persona
class Docente : Persona()

val docente = Docente()
```

Herencia

```
open class Persona(private val nombre: String)
class Docente(private val nombre: String) : Persona(nombre)

val docente = Docente( nombre: "Facundo Rodriguez Arceri")
```

Sobreescribiendo funciones

```
open class Persona {

    open fun saludar(): String {
        return "Hola!"
    }

}

class Docente : Persona() {

    override fun saludar() = "¡Buenos días!"

}
```

Companion objects

- En Kotlin todo es un objeto: no existe el concepto de funciones/métodos estáticos, o de constantes estáticas (que pertenezcan a clases)
- Si se quiere lograr lo mismo, lo que se puede hacer es definir un *companion object* para cada clase

Companion objects

-

```
class Persona {
    companion object {
        fun saludar() = "Hola"
    }
}

fun usandoCompanion() {

    val saludo = Persona.saludar()
    val saludo2 = Persona.Companion.saludar()

}
```

Objetos

Creando un objeto

```
fun crearObjeto() {
    val texto = TextView( context: null)
}
```

Creando un objeto de una clase anónima

```
val listener = object: View.OnClickListener {

    override fun onClick(p0: View?) {
        // Implementación
    }

}
```

Lists

Listas - Usos

- Las listas por default tiene 10 posiciones , cuando se completa, por dentro hace un resize creando un objeto nuevo
- Orden natural que depende de la inserción de elementos a la lista
- Poseen un índice, donde se puede acceder directamente a un determinado elemento
- Proveen una interfaz inmutable (List)

Listas - Creación

Las listas son un conjunto de objetos ordenados. En Kotlin, las listas pueden ser mutables o inmutables

- mutables → *mutableListOf()*
- Inmutables o solo lectura → *ListOf()*

```
val systemUsers: MutableList<Int> = mutableListOf(1, 2, 3)          // 1
val sudoers: List<Int> = systemUsers                                // 2

fun addSystemUser(newUser: Int) {                                   // 3
    systemUsers.add(newUser)
}

fun getSysSudoers(): List<Int> {                                    // 4
    return sudoers
}
```

¿Preguntas?

¿Qué más puede ser útil aprender?

- Colecciones (List, Array, etc)
- Interfaces
- Enums
- Data classes
- Sealed classes
- map(), filter()
- .let(), .apply(), .with(), .also(), etc, etc, etc...

¿Dónde sigo aprendiendo?

https://kotlinlang.org/docs/reference/basic-syntax.html

https://developer.android.com/kotlin

https://play.kotlinlang.org/koans/overview

Fin