

# Desarrollo Android

## Clase 02

# Recapitulamos

# Variables y constantes

Mutables

```
//Variables
var firstName = "John"
var lastName = "Doe"

println("$firstName $lastName") //John Doe

firstName = "Jane"

println("$firstName $lastName") //Jane Doe
```

Inmutables

```
//Constantes
val age = 21
age = 23 //ERROR!
age = calculateAge()

const val code = 10
code = calculateCode() //ERROR
```

# Data types

Numbers

Booleans

Characters

Strings

Arrays

# Data types - Numbers



```
// Enteros (Byte, Short, Int, Long)
```

```
val myAge: Int = 28
```

```
val myAge = 28 //Es equivalente a la línea anterior  
              //El tipo Int es inferido
```

```
val amount: Long = 100
```

```
val amount = 100L //Equivalente a la línea anterior  
                  //Al agregar el sufijo "L", indicamos el tipo de dato
```

```
val amount = 100000000000 //Si el número es lo suficientemente grande  
                           //como para requerir más espacio en memoria  
                           //automáticamente se infiere el tipo de dato Long
```

# Data types - Numbers



```
// Decimales (Float, Double)
```

```
val price: Double = 120.5
```

```
val price = 120.5 //El tipo Double es inferido
```

```
val price: Float = 120.5F //A pesar de que explicitemos el tipo de la variable  
//como Float, el compilador nos obliga a agregar el sufijo F
```

```
val price: Float = 120.12312321312321F //El valor de price se redondea a 120.12312
```

# Data types - Booleans

```
// Booleanos (Boolean)

val isActive: Boolean = true
val isActive = true //El tipo de dato Boolean es inferido

val isUpdated = false

val result = !isActive //NOT

val result = isActive || isUpdated //OR
val result = isActive or isUpdated //OR (no es una evaluación de cortocircuito como ||)

val result = isActive && isUpdated //AND
val result = isActive and isUpdated //AND (no es una evaluación de cortocircuito como &&)

val result = isActive == isUpdated //EQUAL

val result = (isActive && !isUpdated) || (!isActive && isUpdated) //XOR
val result = isActive xor isUpdated //XOR
```

# Data types - Characters



```
// Characters
val character: Char = 'a'
val character2: Char = 'hello' //ERROR

println('b')
println('\u002C')
println('\n')
```



# Data types - Strings



```
// Cadenas de texto
val firstText = "Lorem ipsum dolor"
val secondText: String = "sit amet"

var fullText = firstText + " " + secondText //Lorem ipsum dolor sit amet

fullText = "$firstText $secondText" //Lorem ipsum dolor sit amet
```

# Condicionales



```
// Condicional

var maxValue: Int

if (a < b) {
    maxValue = b
}
else {
    maxValue = a
}
```



```
// Condicional

val maxValue = if (a < b) b else a //Podemos usar if como una expresión

val maxValue = if (a < b) {
    b
}
else if (a == b) {
    Log.v(TAG, "Son iguales") //Podemos usar if como expresión con bloques
    a                        //La última expresión del bloque será el valor del mismo
}
else {
    a
}
```

# Sentencia when



```
// When (símil a switch/case)
```

```
val country = "Uruguay"
```

```
when(country) {  
    "Argentina" -> Log.v(TAG, "Está en Sudamérica")  
    "Uruguay" -> Log.v(TAG, "Está en Sudamérica")  
    "Brasil" -> Log.v(TAG, "Está en Sudamérica")  
    "España" -> Log.v(TAG, "Está en Europa")  
    "Alemania" -> Log.v(TAG, "Está en Europa")  
    else -> Log.v(TAG, "No se ubicarlo")  
}
```

```
when(country) {  
    "Argentina", "Uruguay", "Brasil" -> Log.v(TAG, "Está en Sudamérica")  
    "España", "Alemania" -> Log.v(TAG, "Está en Europa")  
    else -> Log.v(TAG, "No se ubicarlo")  
}
```



```
// When (símil a switch/case)
```

```
val age = 10
```

```
when(age) {  
    0, 1, 2 -> Log.v(TAG, "Es bebé")  
    3..11 -> Log.v(TAG, "Es niño")  
    11..17 -> Log.v(TAG, "Es adolescente")  
    18..69 -> Log.v(TAG, "Es adult@")  
    70..99 -> Log.v(TAG, "Es ancian@")  
    else -> Log.v(TAG, "Es increíble")  
}
```

```
val message = when(age) {  
    0, 1, 2 -> "Es bebé"  
    3..11 -> "Es niño"  
    11..17 -> "Es adolescente"  
    18..69 -> "Es adult@"  
    70..99 -> "Es ancian@"  
    else -> "Es increíble"  
}
```

```
Log.v(TAG, message) //Es niño
```

# Null Safety

```
// Null Safety
```

```
var nullableVariable: String? = null
```

```
nullableVariable.count() //ERROR
```

```
nullableVariable?.count() //CORRECTO (Se evalúa la variable como null, por lo que no se llama a .count())
```

```
nullableVariable!!.count() //CUIDADO! Si bien esto es legal el !! ignora el control de "nulabilidad", por  
//lo que si es null se llama igualmente al .count() dando error en tiempo de ejecución
```

```
var nullableNumber: Int? = null
```

```
val newConstant: Int = nullableNumber ?: 1 //El operador Elvis ?: es un operador binario que devuelve el valor a  
//la izquierda si no es null, y el de la derecha en caso contrario  
//En este caso el valor asignado es 1
```

```
nullableNumber?.let {
```

```
    // Este bloque se ejecuta si nullableNumber NO es nulo
```

```
    // Dentro de este scope, la variable it toma el valor actual
```


```
    // de nullableNumber, y en lugar de ser de tipo Int? es Int (non nullable)
```

```
} ?: run {
```

```
    // Este bloque se ejecuta si nullableNumber SI es nulo
```

```
}
```

# Listas



```
val name = "John"
val lastName = "Doe"
val company = "Google"
val age = "28"

val array = arrayListOf<String>() //Podemos instanciar un ArrayList vacío

list.add(name) //e ir agregando los elementos de a uno
list.add(lastName)
list.add(company)
list.add(age)

val anotherArray = arrayListOf<String>(name, lastName, company, age)
//0 podemos instanciarla directamente con los elementos dentro

val list = listOf<String>(name, lastName, company, age)
//También podemos crear instancias de List en lugar de ArrayList, que
//nos devolverá una instancia de tipo List (por detrás crea un ArrayList)
```

# Arrays



```
// Arrays
```

```
val newList = (1..10).toList()
```

```
val filteredList = newList.filter { it < 5 }
```

```
val sortedList = newList.sortedByDescending { it }
```

```
val firstItemMeetsCondition = newList.find { it in 3..5 }
```

```
Log.v(TAG, filteredList.toString()) //[1, 2, 3, 4]
```

```
Log.v(TAG, sortedList.toString()) //[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

```
Log.v(TAG, firstItemMeetsCondition.toString()) //3
```

# Loops



```
val newList = (1..10).toMutableList()

for (item in newList) {
    Log.v(TAG, item.toString())
}

for (item: Int in newList) {
    Log.v(TAG, item.toString())
}
```



```
val newList = (1..10).toMutableList()

newList.forEach {
    Log.v(TAG, it.toString())
}

for (item in 1..10) {
    Log.v(TAG, item.toString())
}
```

# Maps o Diccionarios



```
//Mapas o Diccionarios
```

```
val firstMap: Map<String, Int> = mapOf()
```

```
val secondMap = mapOf<String, Int>() //Son equivalentes
```

```
val populatedMap = mapOf("John" to 2, "Jane" to 5)
```

```
val mutableMap = mutableMapOf<String, Int>()
```

```
mutableMap.put("John", 2)
```

```
mutableMap["Jane"] = 4
```

```
//populatedMap y mutableMap tienen el mismo contenido
```



# Ejercicio

Crear una lista de personas con los siguientes datos:

- Jorge Perez, 26 años
- Alberto Rodriguez, 30 años
- Norma Pires, 42 años
- Tomás Lorenzo, 12 años
- Alejandra Romero, 29 años
- Norberto Araujo, 68 años
- Omar Junin, 17 años
- Juana Batista, 34 años

A partir de este

- Imprimir los nombres ordenados según la edad (de menor a mayor)
- Imprimir los nombres ordenados según el largo del nombre

# Ejercicio 2

Crear una lista de personas con los siguientes datos:

- Jorge Perez, 26 años, Pintor
- Alberto Rodriguez, 30 años, Maestro
- Norma Pires, 42 años, Ingeniera
- Tomás Lorenzo, 12 años, Estudiante
- Alejandra Romero, 29 años, Abogada
- Norberto Araujo, 68 años, Jubilado
- Omar Junin, 17 años, Estudiante
- Juana Batista, 34 años, Escritora

Por cada persona menor a 18 años imprimir en los logs una frase que diga contenga su nombre y su edad.

Por cada persona de entre 18 y 60 años imprimir en los logs una frase que contenga su nombre, edad y profesión.

Por cada persona mayor a 60 años imprimir en los logs una frase que contenga su nombre y su edad.