Elementos básicos



# Sentencias, bloques y comentarios

#### Sentencias

- En Kotlin, las sentencias se escriben una en cada línea
- No es necesario incluir un ; al final

## Sentencias y;

```
val rocket = "#"; println(rocket)
```

#### Bloques

- Agrupan instrucciones
- Definen el ámbito de las variables
- En Kotlin se utilizan las llaves { y } para delimitarlos

#### Comentarios

- De una línea, //
- De múltiples líneas, /\* \*/
- Se pueden anidar, si están balanceados /\* /\* \*/

# Variables y constantes

#### Declaración

#### Recomendación

- Por defecto, declarar las variables con val
- Cambiarlo a var cuando sea necesario modificar el valor
- Ayuda a evitar errores

# Inferencia de tipos

```
val b = 2
```

## Anotaciones de tipo

```
val c: Int
```

$$c = 3$$

#### Salida por consola

```
println("¡Hola mundo!")

val hora = 9
print("Son las ${hora} en punto")

println(", ¡a cenar!")
```

# Valores numéricos y lógicos

## Tipos de datos numéricos

Tipo	Bits	Descripción
Int	32	Valor numérico entero
Long	64	Valor numérico entero "largo"
Float	32	Valor numérico de precisión simple
Double	64	Valor numérico de precisión doble
Short	16	Valor numérico "corto"
Byte	8	Byte

#### Literales numéricos

	Literal	Notación	Ejemplo
Enteros	Decimal		1_000_000
	Long	L	123L
	Binario	0b	0b1011_1100
	Hexadecimal	0 x	0xF3A
Coma flotante	Double	е	1.25e4
	Float	f o F	123.5f

# Conversiones de tipo

```
val b: Byte = 1 // OK, literals are checked statically
val i: Int = b // ERROR

val i: Int = b.toInt() // OK: explicitly widened
```

## Conversiones de tipo

```
toByte(): Byte
toShort(): Short
toInt(): Int
toLong(): Long
toFloat(): Float
toDouble(): Double
toChar(): Char
```

 Todos los tipos disponen de los métodos de conversión

#### Valores lógicos

```
val orangesAreOrange = true
val turnipsAreDelicious: Boolean = false
```

# Cadenas de texto y caracteres

# String

```
// String
val s = "Hello, world!\n"

// Raw string
val text = """
    for (c in "foo")
        print(c)
"""
```

## Características de los String

- Son inmutables
- Se pueden recorrer los caracteres individuales con un bucle for
- Se pueden comparar directamente con el operador ==

## Plantillas de Strings

```
// Nombre simple
val i = 10
val s = "i = $i" // evaluates to "i = 10"

// Cualquier expresión
val s = "abc"
val str = "$s.length is ${s.length}" // evaluates to "abc.length is 3"
```

#### Caracteres

```
var letra: Char = 'A'
```

# Operadores: asignación y aritméticos

# Operador de asignación

- Copia el contenido de la parte derecha en la parque izquierda
- No devuelve valor
- Hay versiones compuestas, como +=

# Operadores aritméticos

Operador	Operación
+	Suma
	Resta
*	Multiplicación
	División
%	Resto de la división
-i	Menos unario (cambio de signo)
+i	Más unario (no afecta al valor)

# Entrada y salida de datos

#### Salida por consola

```
println("¡Hola mundo!")

val hora = 9
print("Son las ${hora} en punto")

println(", ¡a cenar!")
```

#### Lectura desde teclado

```
// Java
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
String texto = br.readLine();

// Kotlin
val br = BufferedReader(InputStreamReader(System.`in`))
val texto = br.readLine()
```

# Conversión de tipos

## Consulta de tipo

```
if (obj is String) {
    print(obj.length)
if (obj !is String) { // same as !(obj is String)
    print("Not a String")
else {
    print(obj.length)
```

#### Conversiones

	Sintaxis	Notas
Implícitas		
Explícitas	.toInt()	Disponibles para los tipos predeterminados
Smart cast	if(x is String) print(x)	Si el compilador lo puede deducir, no hace falta la explícita
Unsafe cast	as	Excepción si no se puede convertir

# Conversión explícita (casting)

```
val \pi = 3.14159265359 // Double
val pi:Int = \pi // error
val pi:Int = \pi.toInt() // ok
```

#### Smart cast

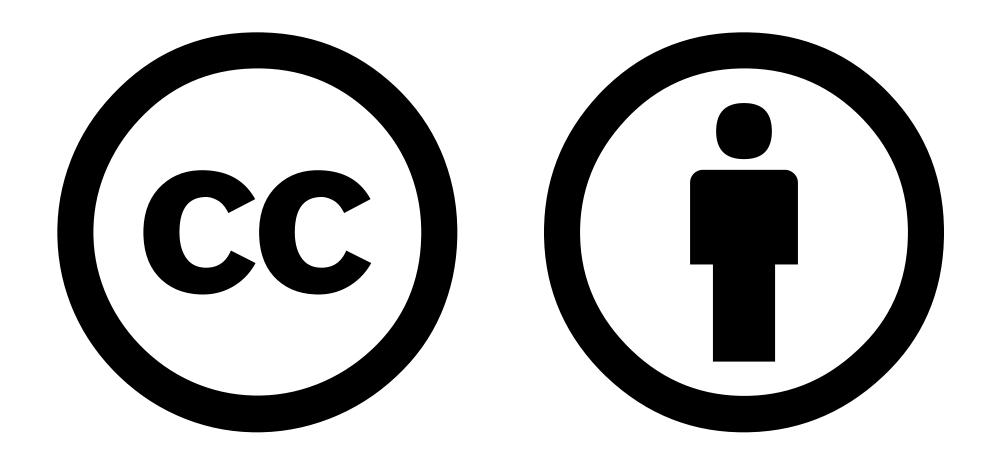
```
fun demo(x: Any) {
    if (x is String) {
        print(x.length) // x is automatically cast to String
    }
}
```

#### Operador de cast

```
val x: String = y as String
   // "unsafe" -> si "y" es null, excepción

val x: String? = y as String?
   // admite null -> no hay excepción

val x: String? = y as? String
   // "safe" -> si "y" es null, no excepción
```



Excepto si se especifica lo contrario, esta presentación está bajo licencia

https://creativecommons.org/licenses/by/4.0/

© 2017 Ion Jaureguialzo Sarasola. Algunos derechos reservados.