Kotlin
Conceptos básicos
Revisión Taller 1

Variables: ¿cómo se declaran?

En Kotlin las variables se declaran de 2 formas:

- var (mutables): las referencias pueden reasignarse a otros objetos.
- val (inmutable, solo lectura): impide reasignar la referencia de un objeto.

```
fun main() {
    val popcorn = 5
    val hotdog = 7
    var customers = 10

    customers = 8
    println(customers)
}
```

var y val controlan referencias no objetos

Tipos de datos

- **Kotlin** es un lenguaje **fuertemente tipado**: el tipo de las variables y expresiones se conoce en tiempo de compilación, sin embargo no requiere que el tipo sea declarado.
- El **compilador** realiza **inferencia de tipos** a partir del contexto de la declaración. La **verbosidad** extra asociada al tipado estático desaparece.

```
fun main() {
    val popcorn = 5.9 //se infiere Double
    val hotdog = 7 //se infiere Int
    var customers = 10 //se infiere Int
    val name= “kiosco” //se infiere String
    customers = 8
    println("En el $name tenemos $customers clientes")
}
```

Las variables se inicializan al declararse o se declaran y luego se inicializan. En este último caso es obligatorio definir el tipo.

```
val c: Int
c = 3 // se inicializa después de declararse
```

Funciones

```
fun sum(x: Int, y: Int): Int {
    return x + y
}
fun main() {
    println(sum(1, 2))
}
```

Valores de default para los parámetros

```
fun printMessageWithPrefix(message: String, prefix: String = "Info") {
    println("[$prefix] $message")
}
```

```
import kotlin.math.PI
fun circleArea(radius: Int): Double {
   return PI * radius * radius
}
fun main() {
   println(circleArea(2))
}
```

Convención de codificación: el nombre de las funciones comienza con inicial minúscula y usa el formato camel-case sin guión bajo.

Clases y objetos

Las **clases** en **Kotlin**, por defecto, son **finales** y **públicas**.

Las **funciones** de las clases, por defecto, son **finales**.

```
class Rectangulo(var ancho: Double, var alto: Double) {
    var perimetro = (ancho + alto) * 2
}
fun main(args: Array<String>) {
    val ancho_local = args[0].toDouble();
    val alto_local = args[1].toDouble();
    val rect = Rectangulo(ancho_local, alto_local)
    println("El perímetro es ${rect.perimetro}")
}
```

```
class Figura {
 fun dibujar() {
     println ("TODO")
 }
}
```

```
class Empty
```

```
abstract class Poligono {
   abstract fun dibujar()
}
```

Constructores

- Una clase en **Kotlin** puede tener un **constructor primario** y uno o más **constructores secundarios**.
- Por defecto, los **constructores** son **públicos**.
- Automáticamente **Kotlin** crea un **constructor** con los parámetros declarados en el encabezamiento de la clase.

```
class Persona (nombre: String) { /*...*/ }
```

```
class Persona constructor(nombre: String) { /*...*/ }
```

```
class Person(val firstName: String, val lastName: String, var isEmployed: Boolean = true)
```

Constructores primarios

No tienen código de inicialización

¿Cómo incluir código de inicialización en el constructor primario?

Usando **bloques de inicialización** con la palabra clave `init`

Constructores

Los **constructores secundarios delegan** en el **constructor primario** directamente o, indirectamente mediante otros constructores secundarios, usando la palabra clave **this**.

Constructores secundarios

Delegación al constructor primario

```
class Persona (val nombre: String) {
    val hijes: MutableList<Persona> = mutableListOf()
    constructor(nombre: String, padre: Persona) : this(nombre) {
        padre.hijes.add(this)
    }
}
```

```
class Constructores {
  init {
        println("Bloque de inicialización")
  }
  constructor(i: Int) {
        println("Constructor secundario $i")
  }
}
```

Herencia

- **Any** es la superclase de todas las clase Kotlin, es **equivalente a Object en JAVA**.
- En **Any** se definen los métodos: equals(), hashCode(), y toString(), disponibles en todas las subclases.
- Las clases en Kotlin son **finales** por lo tanto **no pueden ser heredadas**.
- Para que una **clase sea heredable** es necesario declararla con la palabra clave **open**.

```
open class Base(p: Int)
class Derivada (p: Int) : Base(p)
```

Herencia

- Las **funciones de una clase son finales**, inclusive las declaradas en clases open. Las funciones que se sobreescriben se definen **open**.
- **Kotlin** requiere modificadores explícitos para indicar que un miembro puede ser sobreescrito (**open**) y para indicar que se sobreescribe (**override**).

```
open class Figura {
   open fun dibujar() {
        println ("TODO aún no implementado el dibujar()")
   }
}
```

```
class Rectangulo(var ancho: Double, var alto: Double): Figura() {
      var perimetro = (ancho + alto) * 2
      override fun dibujar() {
         println ("dibujar() de Rectangulo")
       }
}
```

Un miembro declarado con `override` es abierto, por lo que **puede sobreescribirse**. Si se quiere prohibir la reescritura, es necesario declararlo `final`