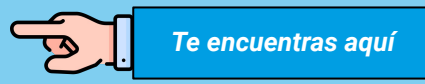
Kotlin para el desarrollo de aplicaciones

Características del Lenguaje Kotlin (Parte I)

{desafío}
latam_

Reconocer las principales características del lenguaje Kotlin para el desarrollo de aplicaciones móviles Android Nativo.

- Unidad 1: Kotlin para el desarrollo de aplicaciones.



- Unidad 2: Ciclo de vida de componentes.
- Unidad 3: Arquitectura en Android.
- Unidad 4: Programación asíncrona en Android.

{desafío} latam_

¿Qué aprenderás en esta sesión?

- *Utilizar la sintaxis del lenguaje Kotlin para la definición de variables, clases y funciones*

Recordemos la clase anterior...

¿Cuál es la diferencia entre Java y Kotlin?

/* Los archivos Kotlin */

Tipos de archivos Kotlin

FILE	CLASS	INTERFACE
Archivo básico con extensión .kt, similar a un archivo de texto plano, la única diferencias es que hace referencia al paquete en que se creó, en este tipo de archivo se pueden agregar clase, enum, funciones, constantes	Clase de Kotlin, este tipo de archivo se ocupa cuando se quiere definir una clase	Pueden contener declaraciones de métodos abstractos, así como implementaciones de métodos. Lo que las diferencia de las clases abstractas es que las interfaces no pueden almacenar estados. Pueden tener propiedades, pero estas deben ser abstractas o proporcionar implementaciones de acceso.

Tipos de archivos Kotlin

DATA CLASS	ENUM CLASS	SEALED CLASS / INTERFACE
Es común crear clases cuyo objetivo principal es almacenar datos. En tales clases, algunas funciones estándar y algunas funciones de utilidad a menudo se derivan mecánicamente de los datos. En Kotlin, se denominan clases de datos	Las clases de enumeración se utilizan para modelar tipos que representan un conjunto finito de valores distintos, como direcciones, estados, modos, etc.	Las clases e interfaces selladas representan jerarquías de clases restringidas que brindan más control sobre la herencia. Todas las subclases directas de una clase sellada se conocen en tiempo de compilación. Ninguna otra subclase puede aparecer fuera de un módulo dentro del cual se define la clase sellada.

/* Variables en Kotlin */

Variables en Kotlin

var

var: es una palabra clave de Kotlin que representa variables mutables no finales.
Una vez inicializadas, somos libres de mutar/cambiar los datos que contiene la variable, por ejemplo:

```
var weight = 50

var list: List<String> = emptyList()
```

variable “weight” con valor de 50

variable “list” definida como una lista de string sin valores

Variables en Kotlin

val

La palabra clave **val** funciona igual que la palabra clave **var**, pero con una diferencia clave: la variable es de solo lectura/no modificable.

Por ejemplo:

```
val user = User()

data class User(
    val firstName: String,
    val pet: Pet,
)
```

crea una instancia de la clase “User”

Variables en Kotlin

lateinit

lateinit significa inicialización tardía. Si no desea inicializar una variable en el constructor, y en su lugar quieres inicializarla más adelante, entonces se puede declarar esa variable con la palabra clave lateinit, pero se debe garantizar la inicialización antes de usarla, no asignará memoria hasta que se inicialice.

```
private lateinit var userViewModel: UserViewModel
```

Se define el tipo, pero no se inicializa

Tip: No se puede utilizar lateinit con val.

Variables en Kotlin

lazy

Lazy: es muy similar a “lateinit”. La variable no se inicializará a menos que use esa variable en el código. Se inicializará solo una vez, después de eso siempre usaremos el mismo valor (no es mutable).

```
val someValue: String by lazy {
        val someLazyValue = "some value"
}
```

Tip: lazy solo puede ser utilizado con val.

{desafío} latam_

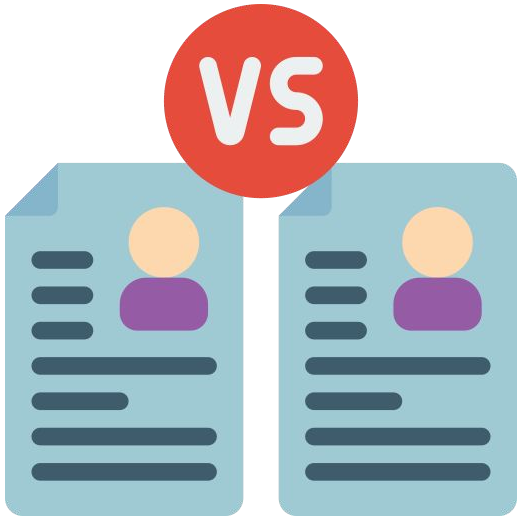
/* Kotlin: Funcional e Imperativo */

¿Que significa que sea imperativo?

Primero debemos entender a qué se refiere cuando dice imperativo:

- La **programación declarativa** es un paradigma de programación que expresa la lógica de un cálculo sin describir su flujo de control.
- La **programación imperativa** es un paradigma de programación que utiliza declaraciones que cambian el estado de un programa.

¿Que significa que sea imperativo?



Otra forma de ver la diferencia entre programación Imperativa y Declarativa es:

- Programación Imperativa se enfoca en “Cómo” obtener el resultado
- Programación Declarativa se enfoca en “Qué” es el resultado que necesito

Imperativa vs Declarativa

- Ejemplo: La siguiente función toma un arreglo de números enteros, multiplica cada uno de los elementos del arreglo por dos, y el resultado lo guarda en otro arreglo.

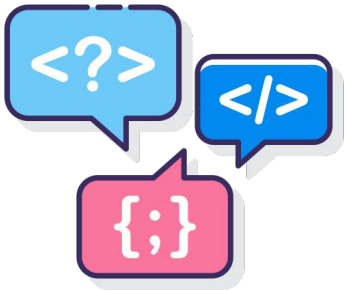
Programación Imperativa

```
fun doubleImperative(numbers: ArrayList<Int>):
ArrayList<Int> {
        val result = arrayListOf<Int>()
        for (i in numbers) {
            result.add(numbers[i - 1] * 2)
        }
        return result
    }
```

Programación Declarativa

```
fun doubleDeclarative(numbers: ArrayList<Int>) =
numbers.map { it * 2 }
```

¿Qué significa que sea funcional?



En programación existen dos paradigmas:

- **Functional programming** - FP (Programación funcional): es un paradigma de programación que escribe funciones puras, lo que significa que estas funciones no modifican las variables, sino que generan otras nuevas como salida.

- **Object Oriented Programming** - OOP (Programación orientada a objetos): es un paradigma de programación que organiza los datos y la estructura del software basándose en el concepto de clases y objetos.

En resumen...


Paradigmas de programación
Declarativo
Imperativo
Lógico
Funcional
Orientado a Objetos
Procedural
Kotlin

¿Qué tiene que ver esto con Kotlin?

Kotlin tiene construcciones **funcionales** y **orientadas a objetos**. Es decir, puedes usar Kotlin como OOP o FP, o mezclar elementos de los dos. Con soporte de primera clase para funciones como funciones de orden superior, tipos de funciones y lambdas, Kotlin es una excelente opción si está haciendo o explorando la programación funcional.

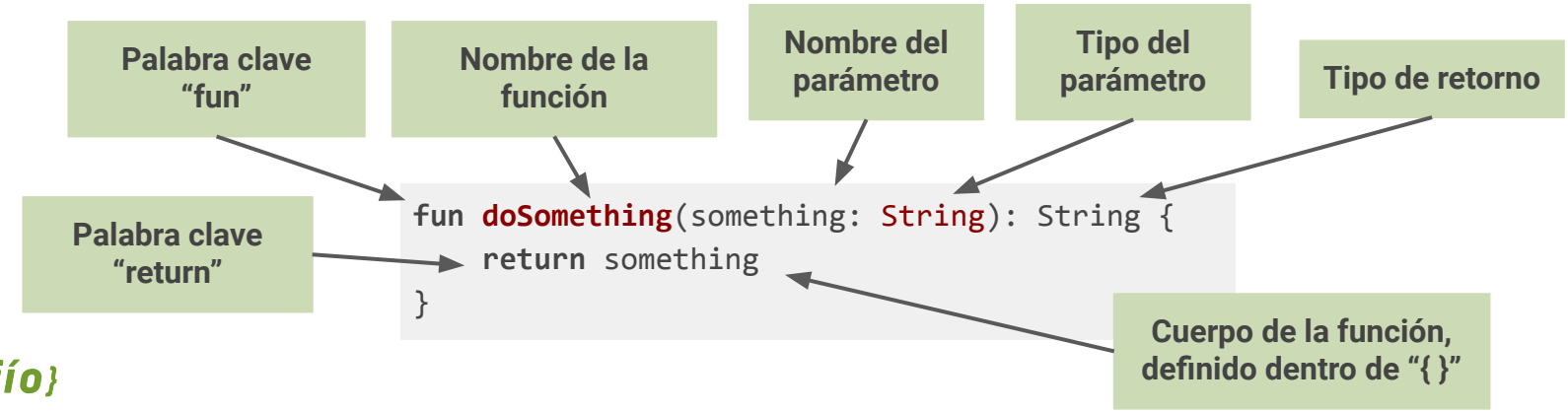
/* Funciones y Clases en Kotlin */

{desafío}
latam_

¿Qué es una función?

- Una **función** es un bloque de código que solo se ejecuta cuando es llamado.
- Se le pueden pasar **datos** o **parámetros**. Estos datos o parámetros son ocupados dentro de la función para realizar alguna operación.
- Una **función** puede o no retornar un resultado.

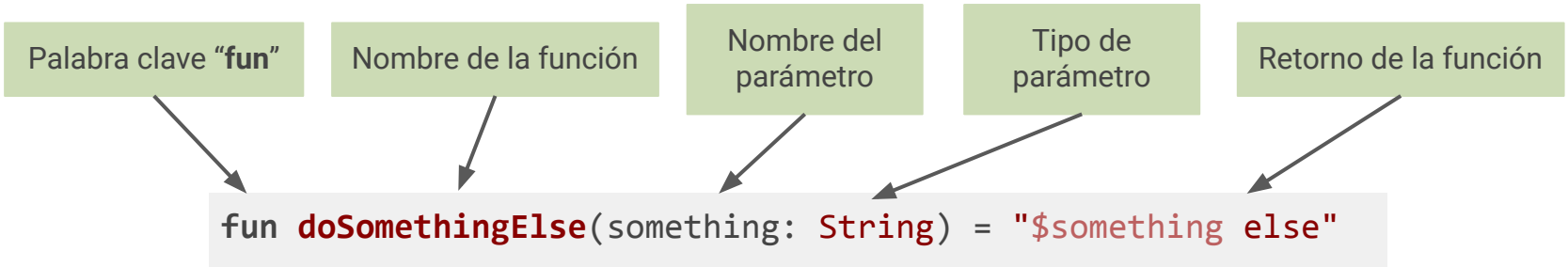
Revisemos un ejemplo:



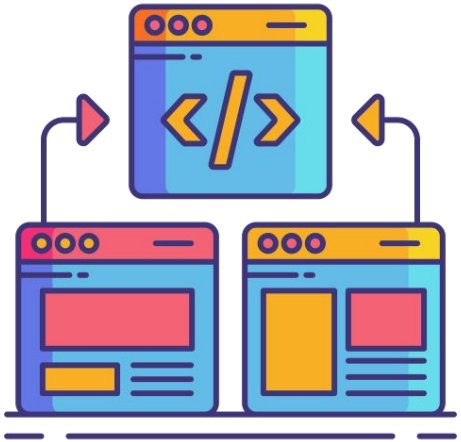
¿Qué es una función?

En Kotlin es común encontrar funciones en la cual no se especifica qué tipo retorna y parece no tener cuerpo de función.

Esta es una forma de escribir la misma función del ejemplo anterior de una forma concisa, sin embargo, no se recomienda para funciones más complejas en las que no se pueda determinar fácilmente qué tipo retorna.



Lambdas y Funciones de Orden Superior



Las funciones de Kotlin son de primera clase, lo que significa que pueden almacenarse en variables y estructuras de datos, y pueden pasarse como argumentos, pudiendo devolverse desde otras funciones de orden superior. Además, puede realizar cualquier operación en funciones que sea posible para otros valores que no sean funciones.

Para facilitar esto, Kotlin, como lenguaje de programación tipificado estáticamente, utiliza una familia de tipos de funciones para representarlas y proporciona un conjunto de construcciones de lenguaje especializadas, como expresiones lambda.

Ejemplo de Lambda

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

val lambda = { number1: Int, number2: Int ->
    sum(number1, number2)
}

fun main() {
    println(lambda.invoke(1, 2))
}
```

Definimos una función simple “suma”

Definimos una variable (val) lambda, con dos parámetros de tipo **Int**, luego en el cuerpo, la variable hace uso de la función **sum**

Finalmente, usamos “invoke” para ejecutar nuestro lambda

¿Qué es una clase?

- Una clase es un modelo para un objeto; comparte propiedades y comportamientos comunes en forma de miembros y funciones de miembros.
- Los objetos se crean a partir de **clases**.

- En Kotlin, una clase se declara con la palabra clave class.

- La declaración de clase consta del nombre de la clase, el encabezado de la clase (que especifica sus parámetros de tipo, el constructor principal, etc.) y el cuerpo de la clase. El cuerpo de la clase se especifica entre dos llaves. Tanto el encabezado como el cuerpo son opcionales. Si la clase no tiene cuerpo, se pueden omitir las llaves.

¿Qué es una clase?

- Los siguientes ejemplos son clases tipo “**data**” y se utilizan para almacenar datos.
- En Android, son ideales cuando se trata de compartir información durante la navegación del usuario entre distintas vistas.

Tip: en Kotlin es posible definir valores por defecto, como en el caso de la clase “Pet” en los campos “id” y “nickName”

```
data class User(
    val firstName: String,
    val pet: Pet,
)

data class Pet(
    val id: Int? = 0,
    val type: PetType,
    val name: String,
    val nickName: String? = null,
)
```

{desafío} latam_

/* Ejecución de una aplicación Kotlin usando la terminal */

Para poder ejecutar una aplicación Kotlin, primero debes tener instalado el compilador. No es común ejecutar aplicaciones Kotlin desde la terminal, por lo general se usan IDE como IntelliJ IDEA o Android Studio.

Instalación manual

Descarga desde el repositorio oficial en GitHub, descomprime el compilador independiente en un directorio y, opcionalmente, agrega el directorio bin a la ruta del sistema.
El directorio bin contiene los scripts necesarios para compilar y ejecutar Kotlin en Windows, macOS y Linux.

Utilizando Brew

Ejecuta:
"`brew update`"
y luego:
"`brew install kotlin`"

Utilizando Snap (Ubuntu)

Ejecuta "`sudo snap install --classic kotlin`"

Una vez terminada la instalación, podemos escribir nuestra primera aplicación, compilarla y luego ejecutarla. Sigue los siguientes pasos:

1. Crea una carpeta en el lugar que sea más fácil acceder desde una terminal

2. En esa carpeta, utilizando cualquier editor de texto, escribe el siguiente código:

```
fun main() {
println("Hola mundo!")
}
```

3. Guarda los cambios y nombra el archivo como "**hello.kt**"

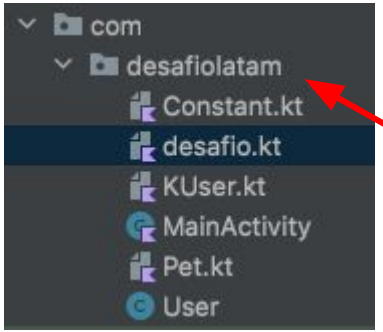
4. Desde la terminal navega hasta la carpeta que contiene el archivo y ejecuta el siguiente comando "`kotlinc hello.kt -include-runtime -d hello.jar`"

   Esto generará un archivo "**hello.jar**", el cual como se puede ver es un archivo java compilado.

5. Finalmente, para ejecutar la aplicación, escribe el siguiente comando "`java -jar hello.jar`"

Kotlin desde la terminal

Otra forma de ejecutar un archivo Kotlin desde la terminal es usando Android Studio



Crea un nuevo archivo dentro tu proyecto Android

Kotlin desde la terminal

Copia y pega el siguiente código en el archivo recién creado:

```
fun main() {
    println("Hello Kotlin!!")
}
```

Notarás el ícono verde que permite ejecutar el archivo, presiónalo y verás el resultado en la terminal dentro de Android Studio.

¡Ahora practiquemos!

Ejercicio

- De manera individual, copia el código a continuación:
- Luego, reemplaza los asteriscos con tus datos y repite los pasos anteriores: guardar, compilar, ejecutar.

```
data class User(
    val name: String,
    val lastName: String,
    val age: Int,
)


fun main() {
    val user = User(
        name = "**************",
        lastName = "**************",
        age = **,
    )

    println("Usuario: nombre: ${user.name}, apellido:
${user.lastName}, edad: ${user.age}")
}
```

Ejercicio

El resultado debería verse así:

```
Usuario: nombre: Nombre, apellido: Apellido, edad: 59
```

¡Felicitaciones, has creado tu primera aplicación en Kotlin!

Intenta responder con tus palabras:

¿Qué es una función?

Intenta responder con tus palabras:

¿Qué es una clase?

Intenta responder con tus palabras:

¿Cuál es la diferencia entre una clase y una interfaz?

De todo lo aprendido el día de hoy ¿Qué me resultó más difícil y por qué?

{desafío} latam_

Próxima sesión...

- *Se desarrollará un **desafío evaluado** en el cual podrás codificar un programa utilizando las sentencias básicas lenguaje Kotlin para resolver un problema simple.*

{desafío}
latam_

Academia de
talentos digitales