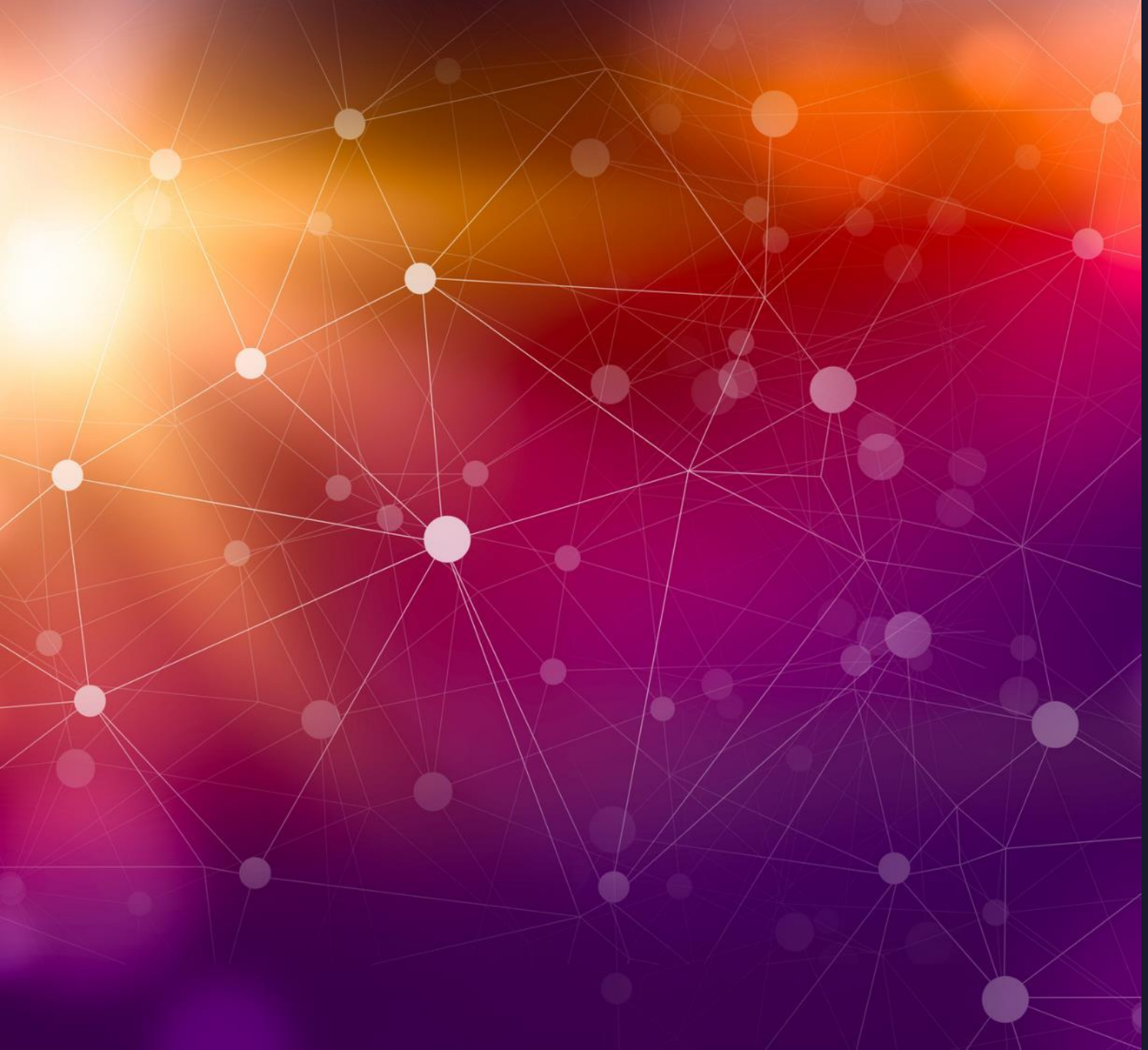

Algunas cosas básicas de Kotlin parte 2

Ilse Suárez

Condicional if

En kotlin if es una expresión, es decir devuelve un valor.

if nos permite ejecutar un bloque de código. Si la condición es verdadera, ejecutará lo implementado en if y si no se cumple ejecuta otra condición o concluirá su proceso.

Si if se usa como expresión para devolver su valor o asignarlo a una variable, la utilización else es obligatoria.

```
fun main() {
    // Condicional if
    val mayorDeEdad = (1..100).random() // Número al azar del 1 al
    100

    if (mayorDeEdad >= 18) println("Es mayor de edad con
    $mayorDeEdad") // Se ejecutará si esta en el rango de 18 al 100.

    val mayorOMenor = if (mayorDeEdad >= 18) "Persona mayor" else
    "Persona menor con $mayorDeEdad"
    println(mayorOMenor)// Ejecutará cualquiera de las 2
    condiciones.

    if (mayorDeEdad >= 41) {
       println("Tiene ganas de descansar")
    } else if (mayorDeEdad in 18..40){
        println("Tiene ganas de trabajar")
    } else {
        println("Tiene ganas de jugar")
    }
}
```

Condicional when

- when define una expresión condicional con varias ramas.
- when se puede utilizar como declaración o como expresión, y si es una expresión la rama else es obligatoria, a menos que se garantice que las ramas definidas cubren todas las posibles opciones.
- El operador in sirve para verificar si un valor está dentro del rango o en una colección.
- El operador is permite comprobar si una variable es de un tipo asignado.

```
fun main() {
    // Condicional when
    print("Ingrese un número del 1 al 10: ")
    val azar: Any? = readLine()?.toIntOrNull() // Me permite
    ingresar un valor en consola.
    when (azar) { // Declaración when
        in 1..3 -> println("El número esta en el rango 1 al 3")
        4,5,6 -> println("El número ingresado esta en los intervalos
        de 4, 5 y 6")
        is String? -> println("Solo esta permitido ingresar números
        enteros")
        in 7..10 -> println("El número esta en el rango 7 al 10")
        !in 1..10 -> println("Número fuera del rango")
    }
}
```

Bucle for

- Es utilizado para iterar cualquier cosa que proporcione un iterador.
- Utiliza el operador in para recorrer los datos estructurados.
- Los datos estructurados pueden ser array, ranges, list, String, etc.

```
fun main() {
    // Bucle for
    for (i: Int in 1 until  5) print(i) // 1,2,3,4

    val nombres = listOf("Armando", "David", "Cely", "Martha")
    for (nombre in nombres.indices) { // Devuelve el índice de cada
         elemento
        if (nombre % 2 == 0) // Solo índices pares
            println(nombres[nombre]) // Armando[0], Cely[2]
    }
}
```

Bucle while

- while comprueba la condición y si cumple ejecuta el cuerpo y a continuación vuelve a la comprobación de condición. Este bucle repetirá su cuerpo mientras la condición sea true o alguna expresión de salto sea evaluada.

```
fun main() {
    var suma = 0
    var valor = 1
    // Bucle while
    while (valor <= 5) {
        suma += valor++
    }
    println("Suma total: $suma") // Suma total: 15
}
```

do-while

- do-while ejecuta el cuerpo y a continuación comprueba el estado. Si está satisfecho el bucle se repite. Por lo tanto, el cuerpo se ejecuta al menos una vez, independientemente de la condición.

```
fun main() {
    // Bucle do-while
    do {
        println("¿Cuál es la capital del Ecuador?")
        println("a. Quito")
        println("b. La Paz")
        println("c. Lima")
        println("Por favor ingresar: a, b o c")
        print("Respuesta: ")
        val opciones = readLine()!! // Ingresa valor en consola.
        val comprobación = opciones != "a"
        if (comprobación) {
            println("Es incorrecto, vuelve a intentarlo! :(")
        } else {
            println("Es correcto, felicitaciones! :)")
        }
    } while (comprobación) // Ejecuta el cuerpo y hace la
      comprobación. Si es incorrecto se repite el bucle, caso
      contrario, termina su proceso.
}
```

Constructor primario

- Forma parte del encabezado de la clase y va tras el nombre de la clase y los tipos de parámetros son opcionales.
- Si el constructor principal no tiene anotaciones ni modificadores de visibilidad, se puede omitir la palabra clave constructor, caso contrario se debe ingresar.
- Las variables val o var en el constructor primario, crea propiedades automáticas que permite ser referenciadas, en un futuro por medio de su tipo de nombre.
- Bloque init es un inicializador adicional que se llama después del constructor principal, para poder contener código y además puede tener uno o varios init.

Constructor secundario

- Puede declararse un constructor secundario, con la palabra clave constructor, en el interior de la clase.
- El cuerpo del constructor secundario se llama después del bloque init.
- Si la clase tiene un constructor primario, cada constructor secundario debe delegar en el constructor primario, ya sea directa o indirectamente a través de otro constructor secundario. La delegación a otro constructor de la misma clase se realiza utilizando la palabra clave: this
- En los parámetros del constructor secundario, no está permitido ingresar variables val y var.

```
// Constructor Primario
class Datos internal constructor(_nombre: String = "Jorge",
_apellido: String) {
    val nombre = _nombre.uppercase()
    val apellido: String = _apellido.uppercase()
    private var edad: Int = 25

    // Bloque init
    init {
        print("Los datos son los siguientes: ") // 1 Ejecutarse
    }

    // Constructor secundario
    constructor(nombre: String, apellido: String, edad: Int) :
    this(nombre, apellido) {
        this.edad = edad
    }

    init {
        print("La edad es: ${edad.inc()}, ") // 2 Ejecutarse
    }
}
```

Object

- Para poder explicarles object es necesario hablar de singleton en Java

```java
public class Singleton {

    private static Singleton instance = null;

    private Singleton(){
    }

    private synchronized static void createInstance() {
        if (instance == null) {
            instance = new Singleton();
        }
    }

    public static Singleton getInstance() {
        if (instance == null) createInstance();
        return instance;
    }
}
```

¿Cuál sería el equivalente en Kotlin?

```kotlin
object Singleton
```

No necesitas más.

Object

- Un object es un tipo de dato con una única instancia estática (Patrón Singleton) y no posee constructores.
- Los object pueden ser expresiones de objeto o declaraciones de objeto y se utilizan con la palabra clave object.
- Las declaraciones object no pueden ser locales (es decir, no se pueden anidar directamente dentro de una función), pero se pueden anidar en otras declaraciones object o clases no internas.
- Las declaraciones object se inicializan, cuando se accede a ellas por primera vez y siempre tiene un nombre después de la palabra clave.
- Las expresiones object se ejecutan (e inicializan) inmediatamente, donde se utilizan y no puede enlazar un nombre después de colocar la palabra clave, pero si es posible heredar de las clases existentes o implementar interface.
- Las expresiones object crean objetos de clases anónimas, es decir, clases que no se declaran explícitamente con la declaración. Tales clases son útiles para un solo uso.

```
// Declaración Object
object CuerpoHumano {
    const val numeroHuesos = 206
    fun print(nombre: String = "Armando"): String {
        return "Hola, mi nombre es $nombre y tengo $numeroHuesos
        huesos."
    }
}

interface CargoDeEmpleo {
    fun cargoTrabajo(cargoEmpleo: String): String
}

fun main {
    // Declaración object
    val cuerpoHumano = CuerpoHumano
    println(cuerpoHumano.print()) // Hola, mi nombre es Armando y
    tengo 206 huesos.

    val datos = object : CargoDeEmpleo { // Expresión Object
        var nombre: String = "Luis"
        var apellido: String = "Orellana"

         override fun toString(): String {
            return "Mi nombre es: $nombre y mi apellido es:
            $apellido."
        }

        override fun cargoTrabajo(cargoEmpleo: String): String {
            return "Mi cargo de trabajo actualmente es:
            $cargoEmpleo."
        }
    }

    println(datos) // Mi nombre es: Luis y mi apellido es: Orellana.
    println(datos.cargoTrabajo("Desarrollador Android")) // Mi cargo
    de trabajo actualmente es: Desarrollador Android.
}
```

Companion object

- Toda clase puede tener un companion object, que es un objeto que es común a todas las instancias de esa clase.
- En los companion object se le puede asignar un nombre, pero no necesario, ya que viene por defecto el nombre Companion y es posible heredar de las clases existentes o implementar interface.
- Los miembros de la clase pueden acceder a los miembros privados del companion object correspondiente.
- Un companion object se inicializa cuando se carga la clase correspondiente.

```
// Implementación companion object
class EmpresaDeCocina {
    companion object : CargoDeEmpleo { // Se puede omitir el nombre
    del companion object, en cuyo caso se utilizará el nombre:
    Companion
        private const val NOMBRE_LOCAL = "Restaurante Mariela"
        override fun cargoTrabajo(cargoEmpleo: String): String {
            return "Trabajo en $NOMBRE_LOCAL, Mi cargo en el trabajo
            es de: $cargoEmpleo."
        }
    }
}

fun main() {
    // companion object
    val empresaDeCocina =
    EmpresaDeCocina.Companion.cargoTrabajo("Concinero") // Se puede
    omitir el nombre Companion
    println(empresaDeCocina) // Trabajo en Restaurante Mariela, Mi
    cargo en el trabajo es de: Concinero.
}
```

Tarea 1 .....

- Interface
- Data class
- Enum class

- Que es
- Ejemplos
- Donde es útil usarlas
- Hay algo parecido en java o se usa igual?
- Y si si existe y es diferente…. Como se usaria? En java

Gracias

Ilse Suárez

Programación de dispositivos móviles

Facultad de Ciencias, UNAM

ilse_suarez@ciencias.unam.mx