Problema 1. Parte teórica. Kotlin

Estudiante: Keyber Yosnar Sequera Avendaño

Carnet: 16-11120

Estructuras de control de flujo en Kotlin:

- Kotlin ejecuta sus programas de forma secuencial de arriba hacia abajo, además no es necesario un operador de secuenciación como el punto y coma (;) al final de la línea de código, lo que hace que su código sea más sencillo de leer y de realizar.
- If-then-else: el equivalente a if-then-else en Kotlin es if-else if-else, que se usa como se muestra a continuación:
- if (condición1) {
- 	// Código a implementar...
- } else if (condición2) {
- 	// Código a implementar...
- } else {
- 	// Código a implementar...
- }

Estructuras de control de flujo en Kotlin:

- Switch-Case: su equivalente en Kotlin es la expresión when que puedan manejar una variedad de casos, incluidos rangos y condiciones complejas, esta expresión luce de la forma:

```
when (opcion) {
    1 -> println("Opción 1 seleccionada")
    2 -> println("Opción 2 seleccionada")
    3, 4 -> println("Opción 3 o 4 seleccionada")
    in 5..10 -> println("Opción entre 5 y 10 seleccionada")
    else -> println("Opción no reconocida")
}
```

- Repetición: Kotlin tiene varias estructuras para realizar ciclos, entre ellas se cuentan:
- Ciclo for: que funciona sobre un rango o sobre una colección:

```
for (i in 1..5) {
    println(i)
}
for (elemento in lista) {
    println(elemento)
}
```

- Ciclo while:

```
while (i < 5) {
    println(i)
    i++
}
```

Estructuras de control de flujo en Kotlin:

- Repetición: continuación:
- Ciclo do-while:
- var j = 0
- do {
- println(j)
- j++
- } while (j < 5)
- También los ciclos pueden usar las palabras claves continue, para saltar un paso en el ciclo o break para romper el ciclo.

- Gestión de excepciones: Kotlin gestiona las excepciones con la estructura try-catch, esta estructura tiene la forma:
- try {
- // Código propenso a excepciones
- } catch (e: Exception) {
- // Manejo de la excepción
- } finally {
- // Código que se ejecutará siempre
- }
- Funciones: Kotlin permite la definición de funciones con la estructura:
- fun nombreFuncion(listaParámetrosDeEntrada): tipoSalida {
- // Código de la función...
- }
- Con las funciones se puede realizar recurrencia en el lenguaje.

Orden de evaluación de expresiones y funciones.

- En general, el orden de evaluación en Kotlin sigue las relas estándar, ya que este lenguaje fue diseñado para que las expresiones se evalúen de manera predecible y lógica según la lógica matemática y de programación. Algunas de estas reglas son:
- Los operadores con mayor precedencia se evalúan primero, la presedencia de los operadores se guía por la precedencia aritmética.
- Los paréntesis pueden modificar el orden de evaluación de las expresiones.
- Los operadores dependiendo de su tipo pueden asociar de izquierda a derecha o de derecha a izquierda.
- Los argumentos de las funciones se evalúan de izquierda a derecha antes de llamar a la función.
- Las expresiones en cortocircuito con los operadores && y || Se evalúan de izquierda a derecha hasta que se conoce el resultado.
- Las funciones y las extensiones se evalúan de izquierda a derecha.

Tipos de datos que posee Kotlin:

- Tipos de datos del lenguaje:
- Tipos numéricos: entre estos cuenta con los tipos Byte, Short, Int, Long, Float, Double.
- Caracteres y cadenas: entre estos cuenta con los tipos Char y String.
- Booleanos: entre estos cuenta con el tipo Boolean (true y false).
- Arreglos: entre estos cuenta con el tipo Array.
- Colecciones: entre ellas se cuentan los tipos List, MutableList, Set, MutableSet, Map, MutableMap
- Tipos nulos: representado por el signo “?” que permite que una variable tenga el valor nulo (null)
- Kotlin además permite la creación de tipos de datos propios mediante clases y objetos:
- Creación de clases en Kotlin: para crear clases en Kotlin se usa la palabra clave class, esta permite la creación de clases nuevas, su estructura es similar a como se muestra a continuación, también hay que mencionar que Kotlin admite herencia de atributos y métodos de otras clases:

Tipos de datos que posee Kotlin:

- class MiClase(parámetro1: tipoParámetro1, ..., parámetroN : tipoParámetroN) : clase1DeLaQueHereda(), ..., claseNDeLaQueHereda() {
- // Propiedades de la clase:
- Propiedad1 = parámetro1
- ...
- PropiedadN = parámetroN
- // Métodos de la clase:
- Fun método1 (parámetro1: tipoParámetro1, ..., parámetroN : tipoParámetroN) {
- // Código del método....
- }
- ....
- Fun métodoN (parámetro1: tipoParámetro1, ..., parámetroN : tipoParámetroN) {
- // Código del método....
- }
- }

Sistema de tipos, tipo de equivalencia para sus tipos y capacidades de inferencia de tipos.

- Sistema de tipos: el sistema de tipos en Kotlin es estático, lo que significa que los tipos de variables y expresiones son verificados en tiempo de compilación. Kotlin utiliza un sistema de tipos fuertemente estático, lo que implica que las operaciones no válidas para un tipo específico se detectan durante la compilación y no en tiempo de ejecución.

- Tipos de equivalencia de tipos:La equivalencia de tipos en Kotlin se rige por los principios de equivalencia nominal y no depende de la estructura interna de los tipos, esto es:
  - Dos tipos se consideran nominalmente equivalentes si tienen el mismo nombre de tipo.
  - Esto significa que aunque dos tipos tengan la misma estructura interna, si tienen nombres diferentes, no son considerados equivalentes nominalmente.

- Capacidades de inferencia de tipos: Kotlin permite al compilador deducir automáticamente el tipo de una variable o una expresión según el contexto y la información disponible. Esta capacidad de inferencia de tipos hace que el código sea más conciso y reduce la necesidad de expecificar tipos de manera explícita, Kotlin puede determinar los tipos automáticamente en expresiones como: val nombreVariable = 42, var nombreVariable = “hola, mundo”, val nombreVariable = listOf(4, 5, 6) (tener en cuenta que los elementos de la lista son del mismo tipo, una lista solo puede tener elementos de un tipo específico, a menos que se declare que admite elementos de tipo null).

Problema 2. Misterio

Estudiante: Keyber Yosnar Sequera Avendaño

Carnet: 16-11120

```
def misterio(a, b, c, d):
0  if c == 0:
1     yield a
2     for x in misterio(b, a, b, d - 1):
3        yield x
4  elif d > 0:
5     for x in misterio(a, b + 1, c - 1, d):
6        yield x


7     a = 2 * X + 3 * Y + 2
8     b = 4 * Y + 5 * Z + 1
9     c = 5 * X + 2 * Z + 3
10    d = (a + b + c) % 7
11    for x in misterio(0, 1, 0, d + 1):
12       print x
```

Global	
a	10
b	9
c	8
d	6
x	0
pc	-11- 12

misterio	
a	0
b	1
c	0
d	7
pc	-0- 1

imprime
0