Curso: Programación móvil

Nombre: Nayelis Belén Velásquez Cruz

Carnet: 2011298

17/05/24

3- Codelabs: Kotlin Basics vale 5 puntos

https://developer.android.com/codelabs/kotlin-bootcamp-basicsç

Answer these questions

Question 1

Which of the following declares an unchangeable list of strings?

- ☐ `val school = arrayOf("shark", "salmon", "minnow")`
- ☐ `var school = arrayOf("shark", "salmon", "minnow")`
- ☐ `val school = listOf("shark", "salmon", "minnow")`
- ☐ `val school = mutableListOf("shark", "salmon", "minnow")`

Question 2

What will be the output of the following code? `for (i in 3..8 step 2) print(i)`

- ☐ 345678
- ☐ 468
- ☐ 38
- ☐ 357

```
fun main() {
    for (i in 3..8 step 2) {
        print(i)
    }
}
```

```
357
```

Question 3

What is the purpose of the question mark in this code? `var rocks: Int? = 3`

- ☐ The type of the variable `rocks` isn't fixed.
- ☐ The variable `rocks` can be set to null.
- ☐ The variable `rocks` cannot be set to null.
- ☐ The variable `rocks` shouldn't be initialized right away.

4- Codelabs: Functions vale 4 puntos

https://developer.android.com/codelabs/kotlin-bootcamp-functions

Question 1

The `contains(element: String)` function returns `true` if the string `element` is contained in the string it's called on. What will be the output of the following code?

```
val decorations = listOf ("rock", "pagoda", "plastic plant", "alligator", "flowerpot")

println(decorations.filter {it.contains('p')})
```

- ☐ `[pagoda, plastic, plant]`
- ☐ `[pagoda, plastic plant]`
- ☐ `[pagoda, plastic plant, flowerpot]`
- ☐ `[rock, alligator]`

Question 2

In the following function definition, which one of the parameters is required? `fun shouldChangeWater (day: String, temperature: Int = 22, dirty: Int = 20, numDecorations: Int = 0): Boolean {...}`

- ☐ `numDecorations`
- ☐ `dirty`
- ☐ `day`
- ☐ `temperature`

Question 3

You can pass a regular named function (not the result of calling it) to another function. How would you pass `increaseDirty( start: Int ) = start + 1` to `updateDirty(dirty: Int, operation: (Int) -> Int)`?

- ☐ `updateDirty(15, &increaseDirty())`
- ☐ `updateDirty(15, increaseDirty())`
- ☐ `updateDirty(15, ("increaseDirty()"))`
- ☐ `updateDirty(15, ::increaseDirty)`

```kotlin
// Definición de la función increaseDirty
fun increaseDirty(start: Int): Int {
    return start + 1
}

// Definición de la función updateDirty que toma una función como argumento
fun updateDirty(dirty: Int, operation: (Int) -> Int): Int {
    // llama a la función "operation" y pasa el valor "dirty" como argumento
    return operation(dirty)
}

fun main() {
    // llamada a la función updateDirty pasando increaseDirty como argumento
    val result = updateDirty(15, ::increaseDirty)
    // Imprime el resultado
    println("Updated dirty: $result")
}
```

```
Updated dirty: 16
```

5- Codelabs: Classes and objects vale 4 puntos

https://developer.android.com/codelabs/kotlin-bootcamp-classes

Answer these questions

Question 1

Classes have a special method that serves as a blueprint for creating objects of that class. What is the method called?

- ☐ A builder
- ☐ An instantiator
- ☐ A constructor
- ☐ A blueprint

Question 2

Which of the following statements about interfaces and abstract classes is NOT correct?

- ☐ Abstract classes can have constructors.
- ☐ Interfaces can't have constructors.
- ☐ Interfaces and abstract classes can be instantiated directly.
- ☐ Abstract properties must be implemented by subclasses of the abstract class.

Question 3

Which of the following is NOT a Kotlin visibility modifier for properties, methods, etc.?

- ☐ `internal`
- ☐ `nosubclass`
- ☐ `protected`
- ☐ `private`

Question 4

Consider this data class: `data class Fish(val name: String, val species:String, val colors:String)`
Which of the following is NOT valid code to create and destructure a `Fish` object?

- ☐ `val (name1, species1, colors1) = Fish("Pat", "Plecostomus", "gold")`
- ☐ `val (name2, _, colors2) = Fish("Bitey", "shark", "gray")`
- ☐ `val (name3, species3, _) = Fish("Amy", "angelfish", "blue and black stripes")`
- ☐ `val (name4, species4, colors4) = Fish("Harry", "halibut")`

```kotlin
// Definición de la clase de datos Fish
data class Fish(val name: String, val species: String, val colors: String)

fun main() {
    // Crear y deconstruir un objeto Fish
    val (name1, species1, colors1) = Fish("Pat", "Plecostomus", "gold")
    val (name2, _, colors2) = Fish("Bitey", "shark", "gray")
    val (name3, species3, _) = Fish("Amy", "angelfish", "blue and black stripes")

    // Imprimir los valores de las propiedades de los objetos Fish deconstruidos
    println("Fish 1: $name1, $species1, $colors1")
    println("Fish 2: $name2, $colors2")
    println("Fish 3: $name3, $species3")
}
```

```
Fish 1: Pat, Plecostomus, gold
Fish 2: Bitey, gray
Fish 3: Amy, angelfish
```

Question 5

Let's say you own a zoo with lots of animals that all need to be taken care of. Which of the following would NOT be part of implementing caretaking?

- ☐ An `interface` for different types of foods animals eat.
- ☐ An `abstract Caretaker` class from which you can create different types of caretakers.
- ☐ An `interface` for giving clean water to an animal.
- ☐ A `data` class for an entry in a feeding schedule.

```kotlin
// 1. Una interface para diferentes tipos de alimentos que comen los animales.
interface Food {
    fun feed()
}

// 2. Una clase abstracta Caretaker de la cual se pueden crear diferentes tipos de cuidadores.
abstract class Caretaker {
    abstract fun clean()
}

// 3. Una interface para dar agua limpia a un animal.
interface WaterProvider {
    fun provideWater()
}
```

6- Codelabs: Extensions vale 4 puntos

https://codelabs.developers.google.com/codelabs/kotlin-bootcamp-extensions/

Answer these questions

Question 1

Which one of the following returns a copy of a list?

- ☐ `add()`
- ☐ `remove()`
- ☐ `reversed()`
- ☐ `contains()`

```kotlin
fun main() {
    val originalList = listOf(1, 2, 3, 4, 5)
    val reversedList = originalList.reversed()

    println("Original list: $originalList")
    println("Reversed list: $reversedList")
}
```

```
Original list: [1, 2, 3, 4, 5]
Reversed list: [5, 4, 3, 2, 1]
```

Question 2

Which one of these extension functions on `class AquariumPlant(val color: String, val size: Int, private val cost: Double, val leafy: Boolean)` will give a compiler error?

- ☐ `fun AquariumPlant.isRed() = color == "red"`
- ☐ `fun AquariumPlant.isBig() = size > 45`
- ☐ `fun AquariumPlant.isExpensive() = cost > 10.00`
- ☐ `fun AquariumPlant.isNotLeafy() = leafy == false`

```
// Definición de la clase AquariumPlant
class AquariumPlant(val color: String, val size: Int, private val cost: Double, val leafy: Boolean)

// Extensiones de la clase AquariumPlant
fun AquariumPlant.isRed() = color == "red"
fun AquariumPlant.isBig() = size > 45
// La siguiente extensión causará un error de compilación debido a que intenta acceder a una propiedad privada
// fun AquariumPlant.isExpensive() = cost > 10.00
fun AquariumPlant.isNotLeafy() = !leafy

fun main() {
    // Creación de una instancia de AquariumPlant
    val plant = AquariumPlant("green", 50, 15.0, true)

    // Llamadas a las extensiones
    println("Is the plant red? ${plant.isRed()}")
    println("Is the plant big? ${plant.isBig()}")
    // println("Is the plant expensive? ${plant.isExpensive()}") // Esta línea causará un error de compilación
    println("Is the plant not leafy? ${plant.isNotLeafy()}")
}
```

```
Is the plant red? false
Is the plant big? true
Is the plant not leafy? false
```

Question 3

Which one of the following is not a place where you can define constants with `const val`?

- ☐ at the top level of a file
- ☐ in regular classes
- ☐ in singleton objects
- ☐ in companion objects

```
// Definición de una constante en el nivel superior de un archivo
const val PI = 3.14159

// Definición de una clase regular con una constante
class MyClass {
    // No se puede usar 'const val' aquí
    val CONSTANT = 10
}

// Definición de un objeto singleton con una constante
object MySingleton {
    const val CONSTANT = 20
}

// Definición de un companion object con una constante
class MyClass2 {
    companion object {
        const val CONSTANT = 30
    }
}
```