Practical class # 8 – Lists in Kotlin

1) Open Kotlin playground, delete the println and create a list

```
val numbers: List<Int> = listOf(1, 2, 3, 4, 5, 6)
```

or using type inference

```
val numbers = listOf(1, 2, 3, 4, 5, 6)
```

print the list and its size

```
println("List: $numbers")
println("Size: ${numbers.size}")
```

Display single elements using the [] notation, the get() function or the first() and last(). Use the function contains() to check if the list contains a specific element.

```
fun main() {
    val numbers = listOf(1, 2, 3, 4, 5, 6)
    println("List: $numbers")
    println("Size: ${numbers.size}")

    // Access elements of the list
    println("First element: ${numbers[0]}")
    println("Second element: ${numbers[1]}")
    println("Last index: ${numbers.size - 1}")
    println("Last element: ${numbers[numbers.size - 1]}")
    println("First: ${numbers.first()}")
    println("Last: ${numbers.last()}")

    // Use the contains() method
    println("Contains 4? ${numbers.contains(4)}")
    println("Contains 7? ${numbers.contains(7)}")
}
```

2) Create a mutableList

Try to create a mutableList

```
val entrees = mutableListOf()
```

Correct the error with

```
val entrees = mutableListOf<String>()
```

or

```
val entrees: MutableList<String> = mutableListOf()
```

Print the elements

```
println("Entrees: $entrees")
```

Add and remove elements to the list

```
fun main() {
    val entrees = mutableListOf<String>()
    println("Entrees: $entrees")

    // Add individual items using add()
    println("Add noodles: ${entrees.add("noodles")}")
    println("Entrees: $entrees")
    println("Add spaghetti: ${entrees.add("spaghetti")}")
    println("Entrees: $entrees")

    // Add a list of items using addAll()
    val moreItems = listOf("ravioli", "lasagna", "fettuccine")
    println("Add list: ${entrees.addAll(moreItems)}")
    println("Entrees: $entrees")

    // Remove an item using remove()
    println("Remove spaghetti: ${entrees.remove("spaghetti")}")
    println("Entrees: $entrees")
    println("Remove item that doesn't exist: ${entrees.remove("rice")}")
    println("Entrees: $entrees")

    // Remove an item using removeAt() with an index
    println("Remove first element: ${entrees.removeAt(0)}")
    println("Entrees: $entrees")

    // Clear out the list
    entrees.clear()
    println("Entrees: $entrees")

    // Check if the list is empty
    println("Empty? ${entrees.isEmpty()}")
}
```

3) Create new lists and loop through them

```
val guestsPerFamily = listOf(2, 4, 1, 3)
var totalGuests = 0
var index = 0
while (index < guestsPerFamily.size) {
    totalGuests += guestsPerFamily[index]
    index++
}
println("Total Guest Count: $totalGuests")

val names = listOf("Jessica", "Henry", "Alicia", "Jose")
for (name in names) {
    println("$name - Number of characters: ${name.length}")
}
```

4) Use classes and list to create a restaurant's menu

```
open class Item(val name: String, val price: Int){
    override fun toString(): String{
        return name
    }
}

class Noodles : Item("Noodles", 10)

class Vegetables : Item("Vegetables", 5)

fun main() {
    val noodles = Noodles()
    val vegetables = Vegetables()
    println(noodles)
    println(vegetables)
}
```

Try to insert more than one vegetable

```
val vegetables = Vegetables("Cabbage", "Sprouts", "Onion")
```

One option to solve the error here is to use

```
class Vegetables(val toppings: List<String>) : Item("Vegetables", 5)

Vegetables(listOf("Cabbage", "Sprouts", "Onion"))
```

Or use the vararg keyword

```
class Vegetables(vararg val toppings: String) : Item("Vegetables", 5)
```

And change the toString according to this definition

```
class Vegetables(vararg val toppings: String) : Item("Vegetables", 5) {
    override fun toString(): String {
        return name + " " + toppings.joinToString()
    }
}
```

Create a complete program to deals with orders.

```
open class Item(val name: String, val price: Int)

class Noodles : Item("Noodles", 10) {
    override fun toString(): String {
        return name
    }
```

```
}

class Vegetables(vararg val toppings: String) : Item("Vegetables", 5) {
	override fun toString(): String {
		if (toppings.isEmpty()) {
			return "$name Chef's Choice"
		} else {
			return name + " " + toppings.joinToString()
		}
	}
}

class Order(val orderNumber: Int) {
	private val itemList = mutableListOf<Item>()

	fun addItem(newItem: Item): Order {
		itemList.add(newItem)
		return this
	}

	fun addAll(newItems: List<Item>): Order {
		itemList.addAll(newItems)
		return this
	}

	fun print() {
		println("Order #${orderNumber}")
		var total = 0
		for (item in itemList) {
			println("${item}: $${item.price}")
			total += item.price
		}
		println("Total: $${total}")
	}
}

fun main() {
	val ordersList = mutableListOf<Order>()

	// Add an item to an order
	val order1 = Order(1)
	order1.addItem(Noodles())
	ordersList.add(order1)

	// Add multiple items individually
	val order2 = Order(2)
	order2.addItem(Noodles())
	order2.addItem(Vegetables())
	ordersList.add(order2)

	// Add a list of items at one time
	val order3 = Order(3)
	val items = listOf(Noodles(), Vegetables("Carrots", "Beans", "Celery"))
	order3.addAll(items)
	ordersList.add(order3)

	// Use builder pattern
	val order4 = Order(4)
```

```
                .addItem(Noodles())
                .addItem(Vegetables("Cabbage", "Onion"))
        ordersList.add(order4)

        // Create and add order directly
        ordersList.add(
                Order(5)
                .addItem(Noodles())
                .addItem(Noodles())
                .addItem(Vegetables("Spinach"))
                )

        // Print out each order
        for (order in ordersList) {
                order.print()
                println()
        }
}
```