COMP 3717 Final Practise

Short Answer

1) Provide an example of a HTTP request

2) What are two benefits of storing your strings in the string resources xml file?

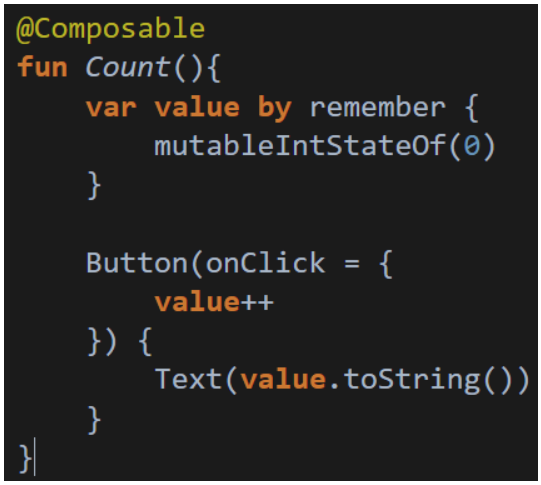
3) What difference does it make when we use delegated properties with our state objects?

4) Is there anything wrong with the code below? Please explain.

```
fun FunctionA() {
    val stateA = remember {
        mutableStateOf("")
    }
}
```

5) Write code to illustrate a callback

6) Make this function stateless using a common pattern with state hoisting



```
@Composable
fun Count(){
    var value by remember {
        mutableIntStateOf(0)
    }

    Button(onClick = {
        value++
    }) {
        Text(value.toString())
    }
}
```

7) What is an example of business logic?

8) What is the difference between `runBlocking` and `launch`?

9) Write code that launches a coroutine and prints "apples and bananas" after a delay of 1 second using async and await

10) Is there anything wrong with this function? If so, how would you fix it without introducing another coroutine builder?

```
suspend fun story(){
    launch {
        delay(1000L)
        println("a time")
    }
    print("Once upon...")
}
```

11) What will be printed to the console?

```
runBlocking{
    var x = 0
    val job = launch {
        repeat( times: 5){
            x++
            println(x)
            delay( timeMillis: 1000L)
        }
    }
    delay( timeMillis: 2500L)
    job.cancel()
}
```

12) Write code to properly model this JSON

```
{
  "id": 1,
  "name": "Taylor",
  "courses": [
    {
      "id": 101,
      "name": "Kotlin Basics"
    },
    {
      "id": 102,
      "name": "Coroutines 101"
    }
  ]
}
```

13) What is the purpose of a key when using `LaunchedEffect`

14) What is the difference between `viewModelScope` and `rememberCoroutineScope`

15) Why will this code not run?

```
fun main() {
    val flow = flowOf(1,2,3,4,5)
    println(flow.toList())
}
```

See next page for answers

Answers

1) Provide an example of a HTTP request?

GET

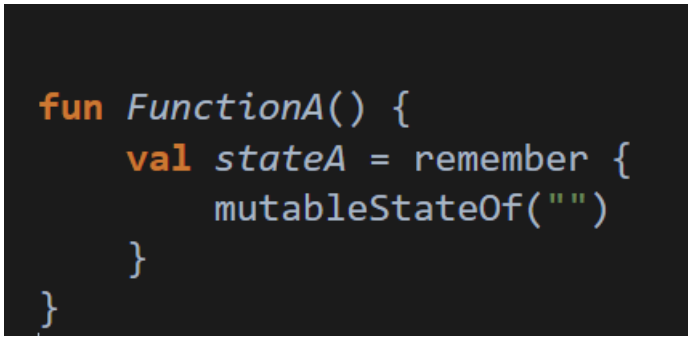
2) What are two benefits of storing your strings in the string resources xml file?

A common and consistent place and for localization purposes

3) What difference does it make when we use delegated properties with our state objects?

We can use delegated properties with our mutable state objects using the *by* keyword instead of the *equal sign*. It allows us to omit using *.value* which makes our code easier to write.

4) Is there anything wrong with the code below? Please explain.



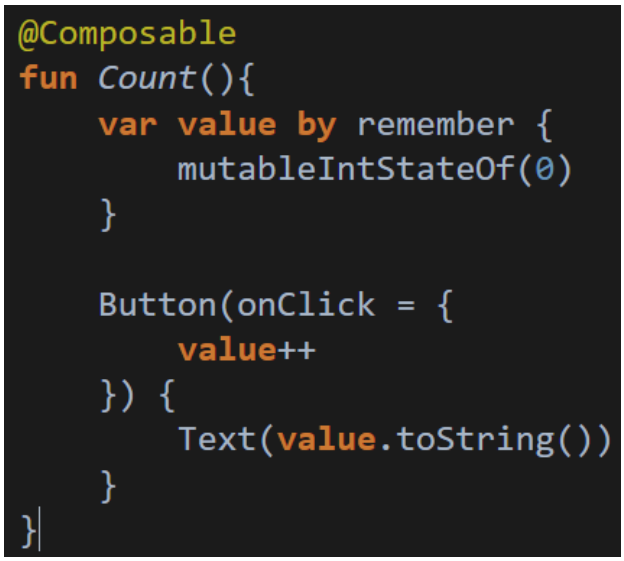
```
fun FunctionA() {
    val stateA = remember {
        mutableStateOf("")
    }
}
```

Yes, remember can only be used in another composable since it is a composable itself.

5) Write some code to illustrate a callback

```
val callback = {
    println("call me later")
}
fun FuncA(callback: ()->Unit){
    callback()
}
```

6) Make this function stateless using a common pattern with state hoisting



```
@Composable
fun Count(){
    var value by remember {
        mutableIntStateOf(0)
    }

    Button(onClick = {
        value++
    }) {
        Text(value.toString())
    }
}
```

```
@Composable
fun Count(value: Int, onValueChanged:() -> Unit) {
    Button(onClick = {
        onValueChanged()
    }) {
        Text(value.toString())
    }
}
```

7) Give me an example of business logic

Checking the validation of an email before it is sent to a database.

8) What is the difference between `runBlocking` and `launch`?

runBlocking is a special type of coroutine builder that is blocking. It shouldn't be used much in production code. Launch is a common non-blocking coroutine builder which launches a coroutine within a provided scope and context.

9) Write code that launches a coroutine and prints "apples and bananas" after a delay of 1 second using *async* and *await*

```
runBlocking {
    val deferred = async {
        delay(1000L)
        "apples and bananas"
    }
    println(deferred.await())
}
```

10) Is there anything wrong with this function? If so, how would you fix it without introducing another coroutine builder?

```
suspend fun story(){
    launch {
        delay(1000L)
        println("a time")
    }
    print("Once upon...")
}
```

```
suspend fun story() {
    coroutineScope {
        launch {
            delay(1000L)
            println("a time")
        }
        print("Once upon...")
    }
}
```

11) What will be printed to the console?

```
runBlocking{
    var x = 0
    val job = launch {
        repeat( times: 5){
            x++
            println(x)
            delay( timeMillis: 1000L)
        }
    }
    delay( timeMillis: 2500L)
    job.cancel()
}
```

```
1
2
3
```

12) Write code to properly model this JSON

```
{
  "id": 1,
  "name": "Taylor",
  "courses": [
    {
      "id": 101,
      "name": "Kotlin Basics"
    },
    {
      "id": 102,
      "name": "Coroutines 101"
    }
  ]
}
```

```
data class User(
    val id: Int,
    val name: String,
    val courses: List<Course>
)

data class Course(
    val id: Int,
    val name: String
)
```

13) What is the purpose of a key when using `LaunchedEffect`

The coroutine will be re-launched if the key changes. If Launched Effect is recomposed and the key did not change, it will not re-launch.

14) What is the difference between `viewModelScope` and `rememberCoroutineScope`

Both provide scope to a coroutine. viewModelScope is used inside a ViewModel and provides scope that is bound to the lifecycle of the Activity. rememberCoroutineScope is used inside a Composable and provides scope that is bound to the lifecycle of the composable.

15) Why will this code not run?

```
fun main() {
    val flow = flowOf(1,2,3,4,5)
    println(flow.toList())
}
```

When working with a flow, any terminal operation such as toList is a suspend function, so it will need to be called inside another suspend function or coroutine.