Practical class # 14 – Threads

1) Open the Kotlin Playground.

2) Create a basic thread

```
fun main() {
    val thread = Thread {
    println("${Thread.currentThread()} has run.")
    }
    thread.start()
}
```

3) Create and run multiple threads

```
fun main() {
    val states = arrayOf("Starting", "Doing Task 1", "Doing Task 2", "Ending")
    repeat(3) {
        Thread {
            println("${Thread.currentThread()} has started")
            for (i in states) {
                println("${Thread.currentThread()} - $i")
                Thread.sleep(50)
            }
        }.start()
    }
}
```

4) Test Unpredictable behaviour

```
fun main() {
    var count = 0
    for (i in 1..50) {
        Thread {
            count += 1
            println("Thread: $i count: $count")
        }.start()
    }
}
```

5) Use a coroutine

```
import kotlinx.coroutines.*

fun main() {
    repeat(3) {
        GlobalScope.launch {
            println("Hi from ${Thread.currentThread()}")
        }
    }
}
```

6) Use a runBlocking

```
import kotlinx.coroutines.*
import java.time.LocalDateTime
import java.time.format.DateTimeFormatter

val formatter = DateTimeFormatter.ISO_LOCAL_TIME
val time = { formatter.format(LocalDateTime.now()) }

suspend fun getValue(): Double {
        println("entering getValue() at ${time()}")
        delay(3000)
        println("leaving getValue() at ${time()}")
        return Math.random()
}

fun main() {
        runBlocking {
        val num1 = getValue()
        val num2 = getValue()
        println("result of num1 + num2 is ${num1 + num2}")
        }
}
```

run the code and then replace the main with

```
fun main() {
        runBlocking {
                val num1 = async { getValue() }
                val num2 = async { getValue() }
                println("result of num1 + num2 is ${num1.await() + num2.await()}")
        }
}
```

7) Rewrite code at point 3 to use coroutines

```
import kotlinx.coroutines.*

fun main() {
        val states = arrayOf("Starting", "Doing Task 1", "Doing Task 2", "Ending")
        repeat(3) {
                GlobalScope.launch {
                        println("${Thread.currentThread()} has started")
                        for (i in states) {
                        println("${Thread.currentThread()} - $i")
                        }
                }
        }
}
```