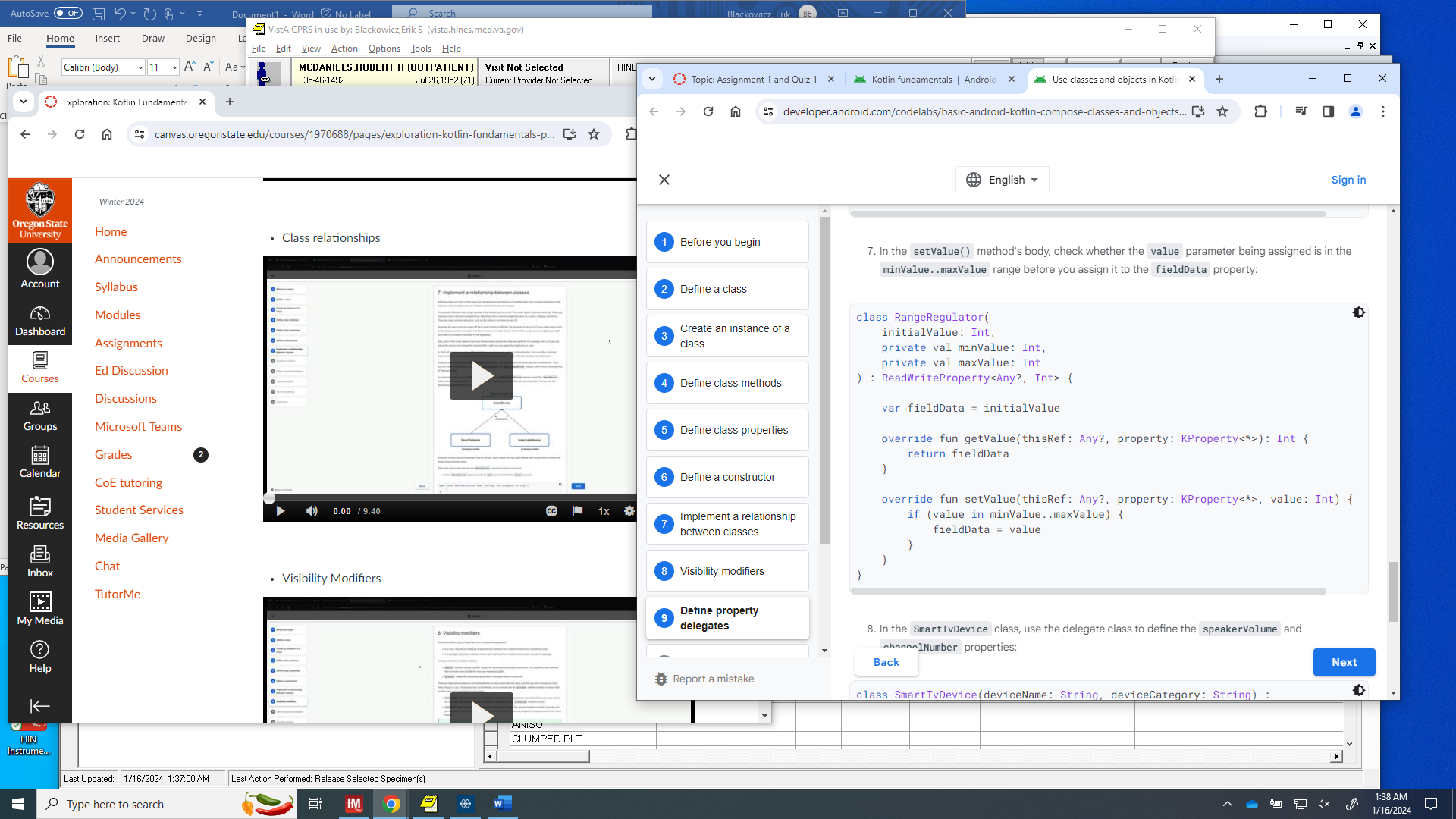
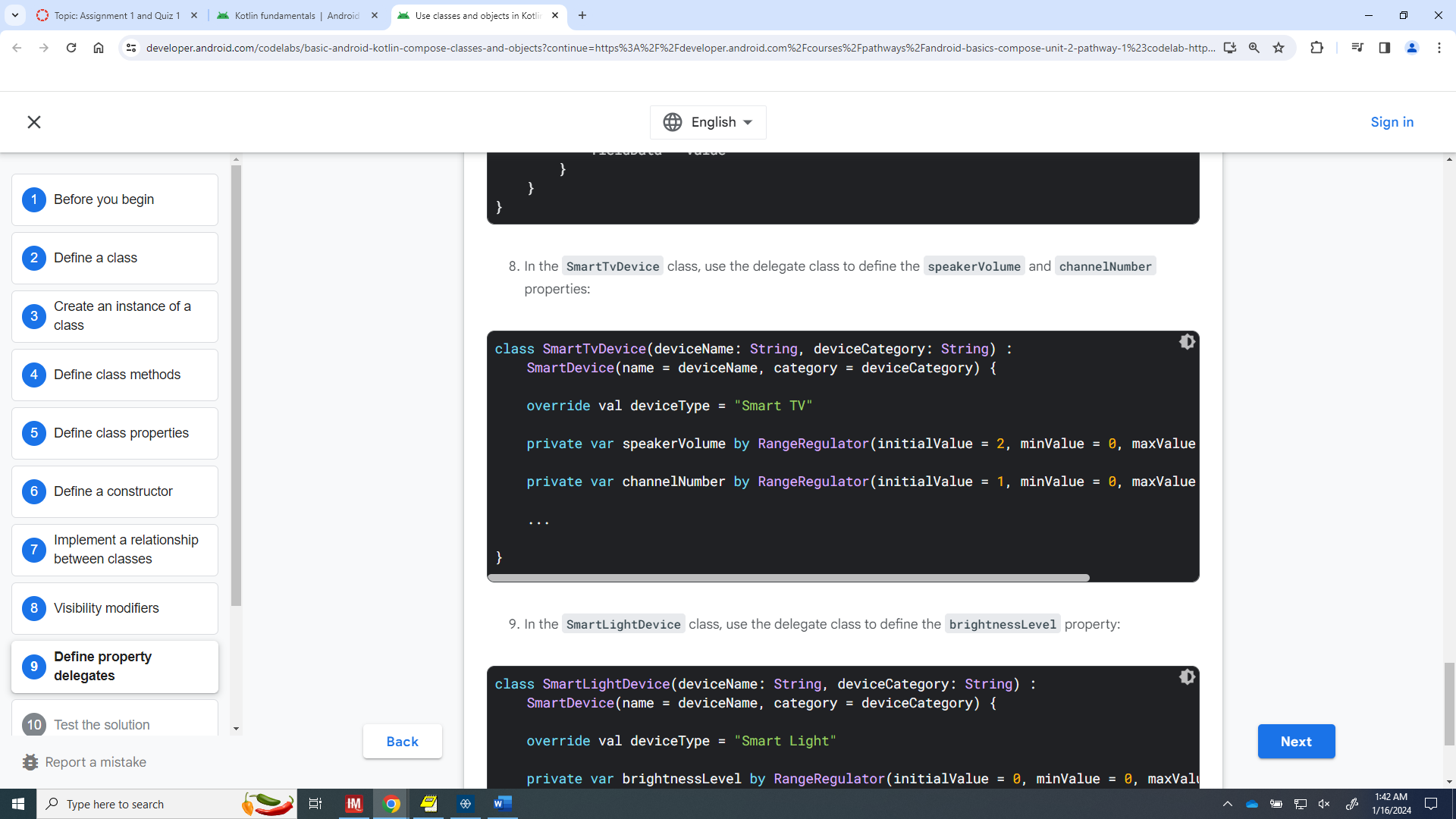
CS 492 Lecture Notes

Module 2 – Part 2- Video 3 – PROPERTY DELEGATES:

Reference material [HERE](https://developer.android.com/codelabs/basic-android-kotlin-compose-classes-and-objects?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-2-pathway-1%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-classes-and-objects#8)

* Property Delegates
* RangeRegulator class is used to limit read,write properties of different data types, I think?
* Example by creating a ‘range regulator’ class to validate, limit input to specific values…
* 
* See above how override is used to rewrite setValue to include check for value in minValue..maxValue
* Also, note thisRef:Any? Is any type of data variable is allowed or Null. Also KProperty is any property type is passed.
* See usage of example above below:
* 

LINK to SMART TV, SMART DEVICE CLASS INHERITANCE Example code [HERE](https://developer.android.com/codelabs/basic-android-kotlin-compose-classes-and-objects?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-2-pathway-1%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-classes-and-objects#9) for reference

Module 2 – Kotline Fundamental Part 3 – Video 1: Functions and Lambda types

Lambda expression = { }

* The idea of Lamda functions is writing functions which returns other functions and their results. Here’s an example of how to store a function in a variable:
  + val trick = {  
        println("No treats!")  
    }
    - This function is stored as a val.
  + val treat: () -> Unit = {  
        println("Have a treat!")  
    }
    - This function is stored as a val, AND explicitly declares the return value as None by using ()-> Unit.
  + Both functions trick() and treat() have the same data type even tho treat() defines it explicitly
* Below is an example of a function which returns a function:
  + fun trickOrTreat(isTrick: Boolean): () -> Unit {  
    }  
      
    val trick = {  
        println("No treats!")  
    }  
      
    val treat = {  
        println("Have a treat!")  
    }

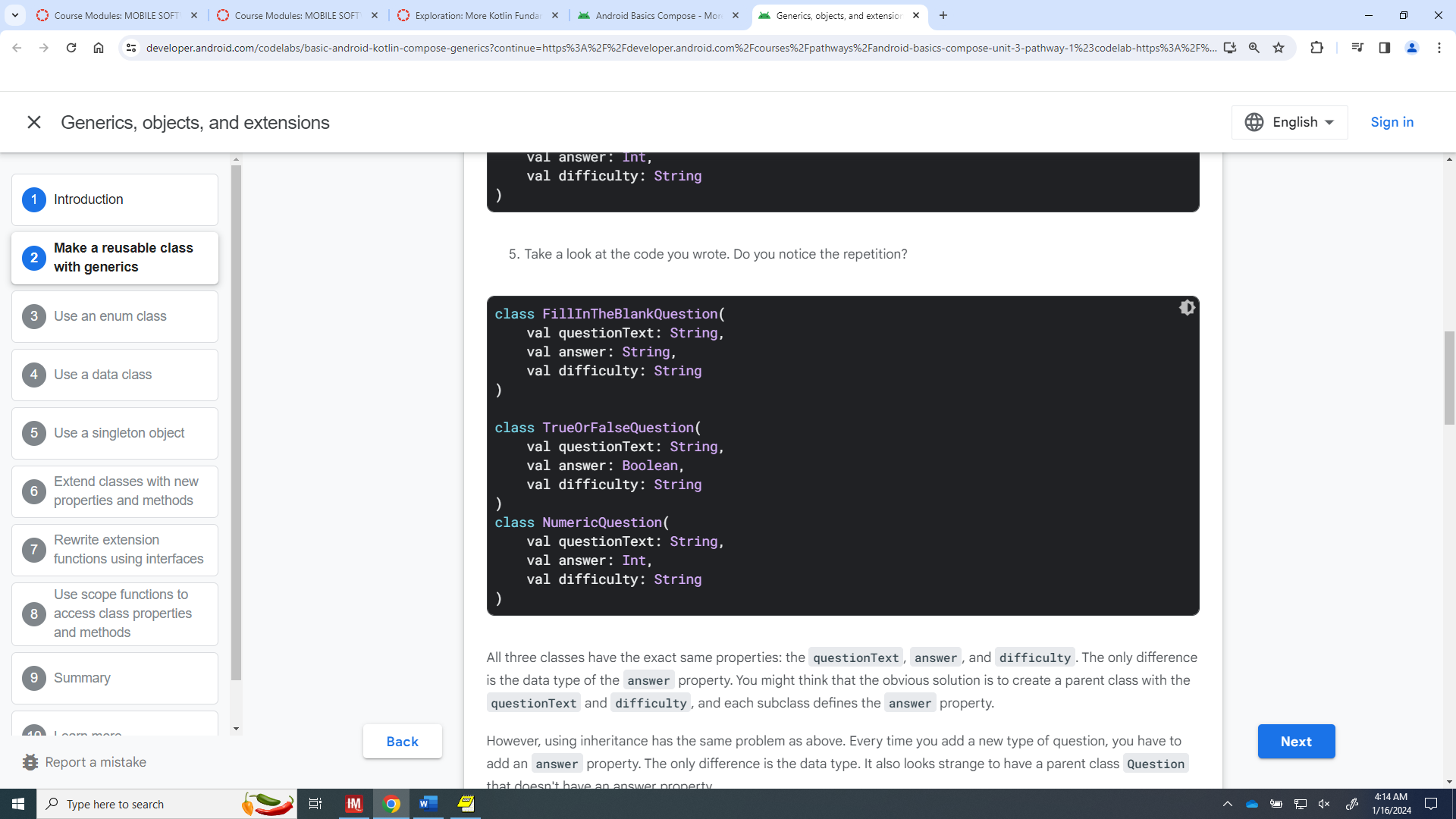
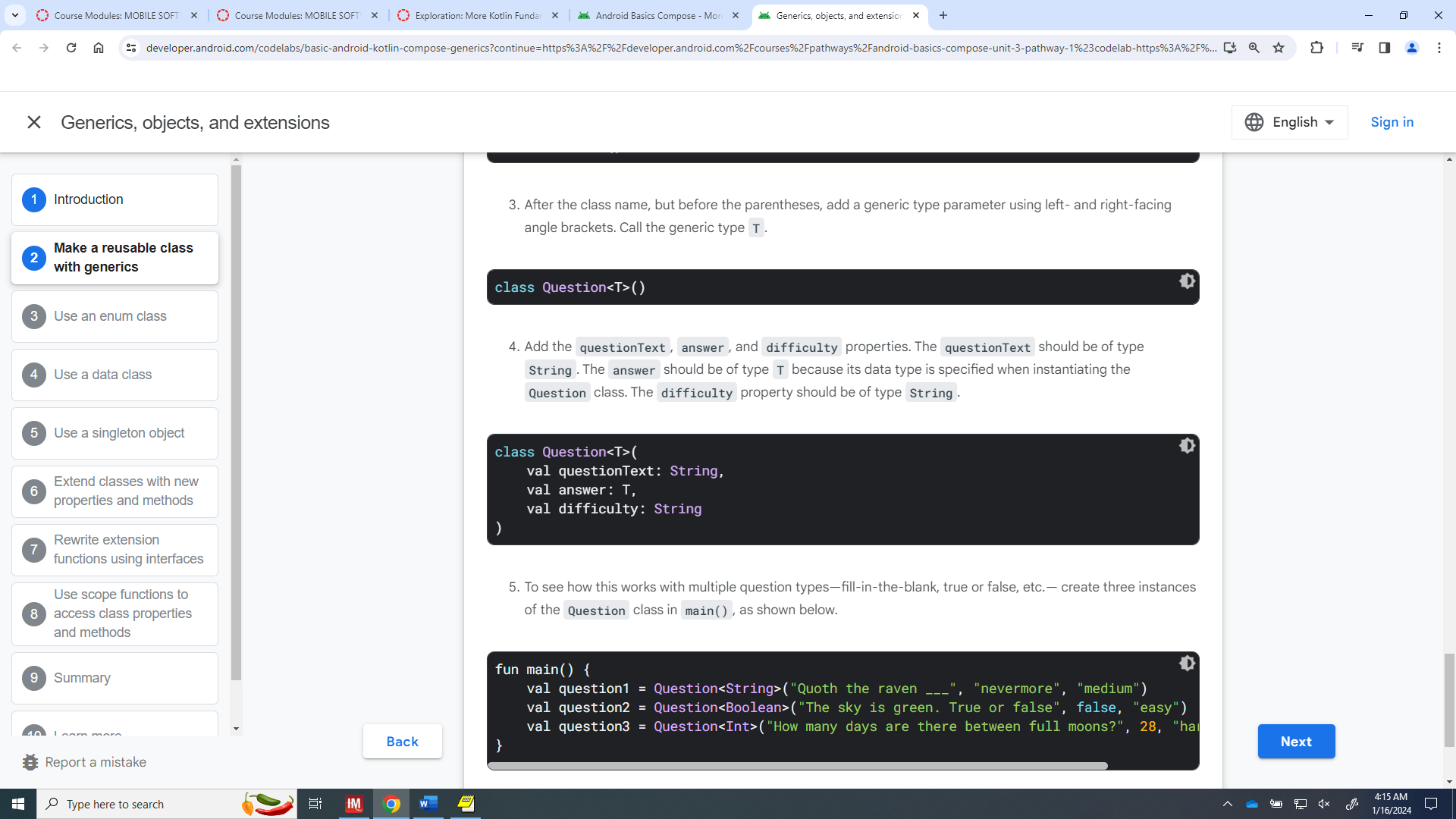
What does the following fun say?:

fun trickOrTreat(isTrick: Boolean, extraTreat: (Int) -> String): () -> Unit {}

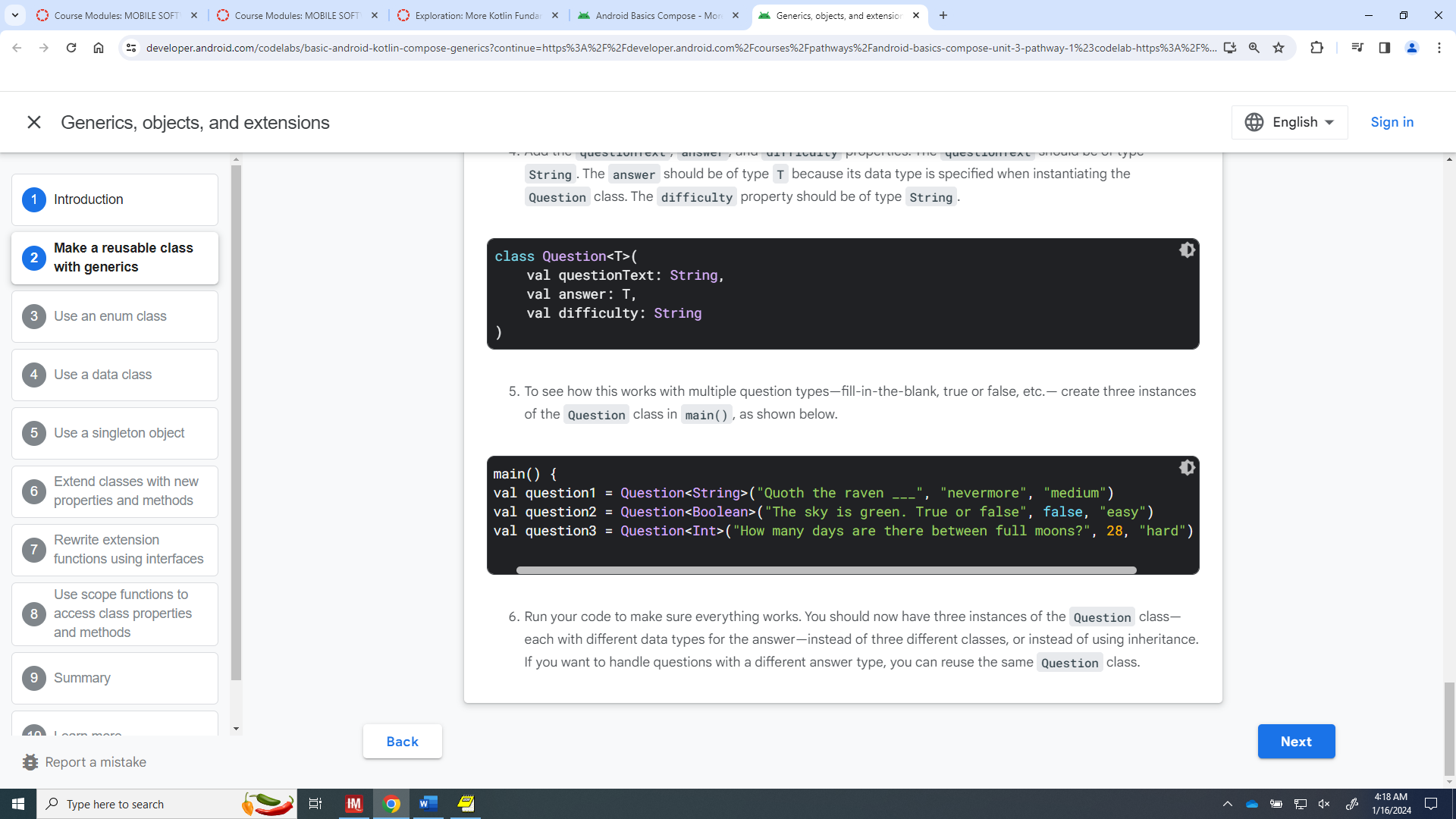
* Function trickOrTreats() has 2 input parameters, #1 is a bool input parameter, #2 is an function input parameter which takes a INT as a parameter which returns a String
* The trickorTreat fx itself returns Nothing, but it does Print something, the trick, treat vals.
* Alternatively, see the code below and notice the nullability of the 2nd parameter(function input):
  + fun trickOrTreat(isTrick: Boolean, extraTreat**: ((Int) -> String)?)**: () -> Unit {  
        if (isTrick) {  
            return trick  
        } else {  
            if (extraTreat != null) {  
                println(extraTreat(5))  
            }  
            return treat  
        }  
    }
* Shorthand syntax using ***it*** keyword:
  + val coins: (Int) -> String = {  
        "$quantity quarters"  
    }
  + Alternatively, could be written seen below, using ***it*** keyword:
  + val coins: (Int) -> String = {  
        "$it quarters"  
    }
* Code along video [HERE](https://developer.android.com/codelabs/basic-android-kotlin-compose-function-types-and-lambda?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-2-pathway-1%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-function-types-and-lambda#1)

Module 3 – More Kotlin Fundamental – Generics – Video 1 – “Generics in Kotlin”:

**Parameterized Type<T>:**  
Can use generic data types to refactor the code below into something better:

 into this...

1. Notice, After the class name, but before the parentheses, add a generic type parameter using left- and right-facing angle brackets. Call the generic type T. The answer should be of type T because its data type is specified when instantiating the Question class.
2. To see how this works with multiple question types—fill-in-the-blank, true or false, etc.— create three instances of the Question class in main(), as shown below.



The <T> is replaced during the instance definition and filled in using various data types.