5COSC023W - Tutorial 2 Exercises - Practice Kotlin

As part of this tutorial for this week, you should complete **ALL** the tasks described in the following specifications: (**make sure that you ask questions to your tutor for anything that you do not understand or if you are stuck at any point**).

You should implement your Kotlin code in one of the Kotlin playgrounds, e.g.:

- `https://play.kotlinlang.org/`

Exercise 1

Define `var x = 1`. Then define `val y = x` and `val z = y`. Next, assign 2 to x and display the values of all three identifiers on different lines: first x, then y, then z.

Try to change the value of `z` to 5 in the end of your code. Why you are not able to change it?

Exercise 2

Create a function `other()` that takes a `String` parameter and returns a `String` containing every other letter of the parameter. For example, the argument "cement" produces "cmn":

```
fun other(s: String): String {
   // ... your code goes here...
}

fun main() {
  println(other("cement"))
}
/* Output:
cmn
*/
```

Hint: The `for` loop in Kotlin accepts a `step` argument which defines the increment value, e.g. `for (i in 1..10 step 3)` will increment the value of `i` by 3 in each iteration.

Execise 3

Implement a function `sum` which accepts 3 double arguments and returns the sum of all its arguments. Test its functionality in `main` by calling the function.

Exercise 4

Write a Kotlin function which accepts a list of numbers as an argument and 2 additional arguments n_1 and n_2. The function creates a new list which contains all the elements of the first list in the range between n_1 and n_2 and returns the new list to the caller.

Test your function by calling it with different lists and arguments.

Hint: A read-only list can be created by using either of the below:

```
var l1 = listOf(1, 3, 5)
var l1 = listOf<Int>(1, 3, 5)
```

A mutable list can be created by using either of the below:

```
var l1 = mutableListOf(1, 3, 5)
var l1 = mutableListOf<Int>(1, 3, 5)
```

The angle brackets denote the type of the elements that the list can contain.

Exercise 5

Implement `everyFifth()` to display every fifth number in the given range. For example, `everyFifth(11, 30)` displays the numbers 15, 20, 25, and 30.

```
fun everyFifth(start: Int, end: Int) {
    // ...
}

fun main() {
  everyFifth(11, 30)
}
/* Output:
15
20
25
30
*/
```

Exercise 6

Implement `everyFifthNonSpace()` to display every fifth non-space character in the given text. For example, `everyFifthNonSpace("abc d e fgh ik")` displays the characters `e` (fifth character if not counting spaces) and `k` (tenth).

```kotlin
fun everyFifthNonSpace(s: String) {
    TODO()
}

fun main() {
    everyFifthNonSpace("abc d e fgh ik")
}
/* Output:
e
k
*/
```

Exercise 7

Write a function that uses a while loop to count the occurrences of a given digit within a decimal number. Place the decimal number in a var called worker. Each pass through the loop tests the right-most digit of worker, then at the end of the loop, removes that right-most digit from worker. The var `occurrences` contains the number of occurrences of the digit you seek.

This table shows the values during each loop while finding occurrences of 1 in 121341:

worker	Removed	occurrences
121341	-	0
12134	1	1
1213	41	1
121	341	1
12	1341	2
1	21341	2
-	121341	3

Hint: What result do you get if you divide an integer n by 10? Which digit do you get if you take the remainder of the division of an integer n by 10? (the remainder (modulo) can be calculated using the percentage sign, i.e. `n % 10` is the remainder of the division of n by 10.

```kotlin
fun countDigits(number: Int, digit: Int): Int {
    var worker = number
    var occurrences = 0
    while (worker > 0) {
        TODO()
    }
    return occurrences
}

fun main() {
    println(countDigits(764241, 4)) // 2
}
```

Exercise 8 - The `Any` Type

The parent type of all types is `Any`.

- You can use it to manipulate objects via variables of the `Any` type.
- However, such a generic variable will not be able to use the specific methods of the specialised type.

1. Run the code below.
2. See what will happen when you attempt to uncomment line 6 and run it again.
3. Comment out line 6 again (i.e. put the whole line as a comment or simply delete it), but uncomment line 12 and run the code again. What happens? Justify this behaviour.

```
fun main() {
   var x: String = "abc"
   println(x.uppercase())

   var y: Any = x
   // println(y.uppercase())  // line 6: does not compile

   var z: Int = 6
   var w: Any = z
   print(w)

   // z = w  // line 12: does not compile
}
```

The `String` object is manipulated via an `Any` variable. When this is the case, we cannot access the methods of the `String` as the compiler cannot check that the variables points to a `String`