**Maja Kałwak**

**Challenge 1**

Zad.1

lastName should be declared as a value (var) and should be equal to something.

Zad.2

Answer 1: true

Answer 2: false

Answer 3: true

Answer 4: true

Answer 5: false

Zad. 3

fun main() {  
 *println*("Enter a number: ")  
 var number = Integer.valueOf(*readLine*())  
 var result: Double = *log2*(number.toDouble())  
 println(result)  
 if (result%2 == 0.0) {  
 *println*("Number is a power of 2")  
 } else {  
 *println*("Number is not a power of 2")  
 }  
}

Zad.4

fun main() {  
 *println*("Enter a number, which should be between 2 and 12: ")  
 var number = Integer.valueOf(*readLine*())  
 while(number<2 || number>12){  
 *println*("Wrong number! Try again: ")  
 number = Integer.valueOf(*readLine*())  
 }  
 val allCombination:Double = 36.0  
 var cube1:Int=1  
 var cube2:Int=1  
 var chance:Int=0  
 for (i in 0..7) {  
 cube1=1  
 for (i in 0..7) {  
 if(cube1 + cube2 == number){  
 chance+=1  
 }  
 cube1+=1  
 }  
 cube2+=1  
 }  
 var prob:Double = chance/allCombination  
 *println*("The probability is equal: " + prob)  
}

**Challenge 2**

Zad.1

The value of sum will be equal 55 and there will be 11 iterations (from 0 to 10, 10 is included).

Zad.2

A Range defines a interval of countable integers. A Range can be closed. It is defined by its two endpoint values which are included or not in the range. By using comments like closedRange.first or closedRange.step we can choose value that we want or choose at what step the interval will be.

Zad.3

Closed range can never be empty because it is defined by its two endpoint values which are both included the range. So even when we create a Range = 0...0. It still contains one element: 0.

Zad.4

When, calls the appropriate piece of code depending on the value given to it. It is similar to the C++ switch function, which also has declared cases inside. In switch, each case needs clearly declared values, e.g. 1, to be able to call further code. When, after entering the value, you can declare cases covering, for example, the whole ranges in which the given number is to fall, to call the corresponding code fragment.

Zad.5

fun main() {  
 for (x in 0..10) *println*(x)  
}

**Challenge 3.1**

Zad.1

a) the function takes String and value and prints the word we choose, as many times as the value we have also entered

b) the function takes value and multiplies it 3 times; function doesn’t work correctly, we have to declare a value, which will be equal (x\*3); example below:

fun main() {  
 fun threeMultiplication(x: Int): Int {  
 var multi = 0  
 multi = x\*3  
 return multi  
 }  
 *print*(threeMultiplication(9))  
}

c) the function takes array and compares values in it and return the max value; we have to for example add Int after args.

fun maxValue(args: Array<Int>): Int

**Challenge 3.2**

Function after filling up dots places:

fun printResult(userChoice: String, gameChoice: String) {  
 val result: String  
//Figure out the result  
 if (userChoice==gameChoice) result = "Tie!"  
 else if ((userChoice=="Rock" && gameChoice=="Scissors")||  
 (userChoice=="Paper" && gameChoice=="Rock")||  
 (userChoice=="Scissors" && gameChoice=="Paper")) result = "You win!"  
 else result = "You lose!"  
 *println*("You chose $userChoice. I chose $gameChoice. $result")  
 }

**Challenge 4**

Zad.1

Nullable types - boxes that may contain a null value or a ‘normal’ value

Null Safety - Kotlin is a language that tries to prevent the creation of a null reference while removing the possible dangers associated with this reference. The compiler will not allow you to assign a null to a reference unless we let it know about it. We can do it be using ‘?’.

Zad.2

Elvis operator it is used when you want to get a value out of the nullable. If that value is null, then you’ll use a default value. It is a way of making a null-check and provide a default value.

Zad.3

?. can be used to return or throw a different kind of exceptions or returns null if the property is null. It is a safety way to check if the reference stores null

!! It is not that safety. It will only throw a null pointer exception.

Zad.4

In var age after Int should be ? 🡪 var age: Int? = null

In var distance, should be type Double 🡪 var distance: Double = 27.6

Zad. 5

fun devideIfWhole (a : Int, b : Int) : Int? {  
 var howmany: Int? = null  
 val validRestOfDivision: Double = 0.0  
 var currentDividend = a  
 var restOfDivision: Double = 0.0  
 var flag: Boolean = true  
 when(b){  
 0 -> *println*("It isn't possible to divide")  
 1 -> *println*("Infinity")  
 else -> flag = false  
 }  
 while(!flag){  
 restOfDivision = (currentDividend%b).toDouble()  
 currentDividend /= b  
 if (restOfDivision==validRestOfDivision) {  
 howmany = howmany?:0  
 howmany = howmany +1  
 }else{  
 flag = true  
 }  
 }  
 return howmany  
}

. **Challenge 5**

Zad.1

fun removeOne(item: Int, list: List<Int>): List<Int> {  
 var lista = list.*toMutableList*()  
 lista.remove(item)  
 return lista  
}

Zad.2

fun remove(item: Int, list: List<Int>): List<Int> {  
 var lista = list.*toMutableList*()  
 var x = 0  
 while(x < lista.size) {  
 var num = lista[x]  
 if(num == item){  
 lista.remove(item)  
 }  
 x = x + 1  
 }  
 return lista  
}

Zad.3

fun reverse(array: Array<Int>): Array<Int> {  
 val arrayLength: Int = array.size  
 val zeroArray = Array(arrayLength) **{** 0 **}** var newIndeks: Int = 0  
 for (x in arrayLength - 1 *downTo* 0) {  
 zeroArray[newIndeks] = array[x]  
 newIndeks = newIndeks + 1  
 }  
 return zeroArray  
}

Zad.4

fun main() {  
 fun maxminValue(args: Array<Int>) {  
 var max = args[0]  
 var min = args[0]  
 var x = 1  
 while (x < args.size) {  
 var item = args[x]  
 max = if (max >= item) max else item  
 min = if (min <= item) min else item  
 x = x + 1  
 }  
 return *print*("Max: " + max + "\nMin: " + min )  
 }  
 maxminValue(*arrayOf*(-1, 9, 3, 4))  
}

**Challenge 6**

Zad.1

Lambda is a function, which has no name. To use lambda, we have to assign it to a variable or constant. We cannot names arguments in lambda expression, because it will be treated like an error.

Zad.2

‘it’ it is a name of a single parameter in lambda.   
It allows for a shorter and more convenient entry if you want to perform the same operation on all the given arguments.

Zad.3

fun repeatTask(times: Int, task: (() -> Unit)?) {  
 if (times > 0) {  
 var x = 0  
 while (x < times) {  
 task?.*let* **{ it**() **}** x = x + 1  
 }  
 }  
}

Zad.4

fun main() {  
 val appRatings = *mapOf*(  
 "KalendarzPro" *to arrayOf*(1,5,5,4,2,1,5,4),  
 "Kurier Messenger" *to arrayOf*(5,4,2,5,4,1,1,2),  
 "Myszojeleniopedia" *to arrayOf*(2,1,2,2,1,2,4,2)  
 )  
 var averageRatings = *mutableMapOf*<String,Double>()  
 appRatings.*forEach* **{** (name,ratings) **->** averageRatings[name] = ratings.*reduce***{** sum, element **->** sum+element**}**/ratings.size.toDouble() **}** var averageGreaterThanThree = (averageRatings.*filter* **{** (name,average) **->** average > 3.0 **}**).*map***{it**.key**}** *print*(averageGreaterThanThree)  
}