How to write functions?

A function is a block of code that only runs when it is called. You can pass data, known as parameters, into a function. Functions are used to perform certain actions, and they are also known as methods.

There are some effective tips to follow while writing functions. They are

1. Functions should be small.
2. Block and indenting
3. Do one thing.
4. Function names should say what they do
5. Avoid Side Effects
6. Remove dead code
7. Low number of arguments

1. Functions should be small.

The function length must not exceed 20 lines.

For Example:

Bad Practice:

```
fun filterMultiples(limit:Int):List<Int>{
    var multiples:MutableList<Int> = mutableListOf()
    for(iterator in 1 ≤ .. ≤ limit)
    {
        if(iterator%3 ==0 )
        {
            multiples.add(iterator)
        }
        else if(iterator%5 == 0)
        {
            multiples.add(iterator)
        }
    }
    return multiples
}
```

Good Practice:

```
fun filterMultiple(limit: Int): List<Int> {
    return (1 ≤ .. ≤ limit).filter { it % 3 == 0 || it % 5 == 0 }
}
```

2. Block and indenting

Don't use long *if-else, switch,* and *while* statements, because sometimes they make functions too big and messy.

Bad Practice:

```
fun main()
{
    var count:Int =0
    var prime:Int =0
    var number : Int =2
    val limit = 10001
    while(count != limit)
    {
        var flag :Boolean = true
        for (i in 2 ≤ .. ≤ number / 2) {
            if (number % i == 0)
                flag = false
        }
        if(flag) {
            prime = number
            count++
        }
        number++
    }
    println(prime)
}
```

Good Practice:

```
fun Int.isPrime():Boolean
{
    for (i in 2 ≤ .. ≤ this / 2) {
        if (this % i == 0)
            return false
    }
    return true
}
fun main()
{
    var count:Int =0
    var prime:Int =0
    var number : Int =2
    val limit = 10001
    while(count != limit)
    {
        if(number.isPrime()) {
            prime = number
            count++
        }
        number++
    }
    println(prime)
}
```

3. Do one thing.

The function must do only one thing correctly.

Bad Practice:

```
fun LargestPrimeFactor(number:Long): Long{
    var largePrime:Long = 0
    var flag :Boolean = true
    for(i in 3L ≤ .. ≤ 10000L step 2)
    {
        for (j in 2 ≤ .. ≤ i / 2) {
            if (i % j == 0.toLong())
                flag = false
        }     // checking whether prime
        if(number%i == 0.toLong() && flag) // checking whether factor of given number
            largePrime =i
    }
    return largePrime
}
fun main()
{
    var number:Long = 600851475143
    println(LargestPrimeFactor(number))
}
```

Good Practice:

```
fun Long.isPrime():Boolean
{
    for (i in 2 ≤ .. ≤ this / 2) {

        if (this % i == 0.toLong())
            return false
    }
    return true
}
fun LargestPrimeFactor(number:Long): Long{
    var largePrime:Long = 0
    for(i in 3L ≤ .. ≤ 10000L step 2)
    {
        if(number%i == 0.toLong() && i.isPrime())
            largePrime =i
    }
    return largePrime
}
fun main()
{
    var number:Long = 600851475143
    println(LargestPrimeFactor(number))
}
```

4. Function names should say what they do

Naming the function is important, the function name must tell what is happening inside the function.

Bad Practice:

```
fun Long.function():Boolean
{
    for (i in 2 ≤ .. ≤ this / 2) {

        if (this % i == 0.toLong())
            return false
    }
    return true
}
```

Good Practice:

```
fun Long.isPrime():Boolean
{
    for (number in 2 ≤ .. ≤ this / 2) {

        if (this % number == 0.toLong())
            return false
    }
    return true
}
```

5. Avoid Side Effects

The function should not affect the global variables or some other factors in that program.

Bad Practice:

```
var limit = 100
fun findEven() : MutableList<Int>
{
    var evenNumbers : MutableList<Int> = mutableListOf()
    while(limit>0)
    {
        if(limit%2 == 0)
            evenNumbers.add(limit)
        limit-- // limit is affected here
    }
    return evenNumbers
}
fun main()
{
    var evenNumbers :MutableList<Int> = findEven()
    println("Even Numbers upto $limit")
    println(evenNumbers)
}
```

Good Practice:

```
var limit = 100
fun findEven() : MutableList<Int>
{
    var end = limit // here the limit value is not affected
    var evenNumbers : MutableList<Int> = mutableListOf()
    while(end>0)
    {
        if(end%2 == 0)
            evenNumbers.add(end)
        end--
    }
    return evenNumbers
}
fun main()
{
    var evenNumbers :MutableList<Int> = findEven()
    println("Even Numbers upto $limit")
    println(evenNumbers)
}
```

6. Remove dead code

Unused code can be removed to make the code look clean.

Bad Practice:

```
fun Long.isPrime():Boolean
{
    for (i in 2 ≤ .. ≤ this / 2) {

        if (this % i == 0.toLong())
            return false
    }
    return true
    println("This returns true") // this is unused code
}
```

7. Low number of arguments

The function should not contain more than 3 arguments. If it is make them a class and use them as a single argument.

Bad Practice:

```
fun findVolume(length:Int, breadth:Int , height:Int):Int
{
    return(length*breadth*height)
}
```

Good Practice:

```
class Cuboid(var length:Int, var breadth:Int, var height:Int)
{


}
fun findVolume(cuboid: Cuboid ):Int
{
    return(cuboid.length*cuboid.breadth*cuboid.height)
}
```