Assignment Topic:

Functions Assignment Level Basic

B1. WAP to find max value using function as express

```
fun main(args: Array<String>) {
 val a = 33
 val b = 122
 val max = max(a,b)
 println(max)
}
fun max(a:Int,b:Int) : Int = if (a>b) a else b
```

B2. Write simple function name start() return the string “OK” (basic function)

```
fun main(args: Array<String>) {
 val max = start()
 println(max)
}
fun start() = "Ok"
```

B3. WAP to multiply two number with validation if number if not null. (not null)

```
fun main(args: Array<String>) {
 val a:Int?= 3
 val b:Int?= 2
 val max = multi(a!!, b!!)
 println(max)
}
fun multi(a:Int,b:Int) :Int = a*b
```

B4. Kotlin Program to Display Prime Numbers Between Intervals Using Function

```
fun main(args: Array<String>) {
   var low = 30
```

```
    val high = 50
    while (low < high) {
        if (checkPrime(low))
            println(low.toString())
        ++low
    }
}
fun checkPrime(num: Int): Boolean {
    var boolean = true
    for (i in 2..num / 2) {
        if (num % i == 0) {
            boolean = false
            break
        }
    }
    return boolean
}
```

B5. Kotlin Program to Display Armstrong Numbers Between Intervals Using Function

```
fun main(args: Array<String>) {
    val low = 1
    val high = 100
    for (number in low + 1..high - 1) {
        if (checkArmstrong(number))
            print("$number ")
    }
}
fun checkArmstrong(num: Int): Boolean {
    var digits = 0
    var result = 0
    var originalNumber = num
    while (originalNumber != 0) {
```

```
        originalNumber /= 10
        ++digits
    }
    originalNumber = num
    while (originalNumber != 0) {
        val remainder = originalNumber % 10
        result += Math.pow(remainder.toDouble(), digits.toDouble()).toInt()
        originalNumber /= 10
    }
    if (result == num)
        return true
    return false
}
```

B6. Kotlin Program to Check Whether a Number can be Expressed as Sum of Two Prime Numbers

```
fun main(args: Array<String>) {
    val number = 34
    var flag = false
    for (i in 2..number / 2) {
        if (checkPrime(i)) {
            if (checkPrime(number - i)) {
                System.out.printf("%d = %d + %d\n", number, i, number - i)
                flag = true
            }
        }
    }
    if (!flag)
        println("$number cannot be expressed as the sum of two prime
numbers.")
}
fun checkPrime(num: Int): Boolean {
    var isPrime = true
```

```
    for (i in 2..num / 2) {
        if (num % i == 0) {
            isPrime = false
            break
        }
    }
    return isPrime
}
```

B7. Kotlin Program to Find the Sum of Natural Numbers using Recursion

```
fun main(args: Array<String>) {
 val a:Int = 1
 println(sumofnaturalnumber(1))
}
var sum:Int = 0
fun sumofnaturalnumber(num:Int): Int {
 sum = sum +num
  if (num >= 100) {
  return sum
 } else return  sumofnaturalnumber(num + 1)
}
```

B8. Kotlin Program to Find Factorial of a Number Using Recursion

```
fun main(args: Array<String>) {
 val a:Int = 1
 println(factorial(3))
}
fun factorial(num:Int): Int {
 if(num <=1){
  return 1
 }
```

```
 else return num*factorial(num-1)
}
```

B9. Kotlin Program to Find G.C.D Using Recursion

```
fun main(args: Array<String>) {
    val n1 = 366
    val n2 = 60
    val hcf = hcf(n1, n2)
    println("G.C.D of $n1 and $n2 is $hcf.")
}
fun hcf(n1: Int, n2: Int): Int {
    if (n2 != 0)
        return hcf(n2, n1 % n2)
    else
        return n1
}
```

B10. Kotlin Program to Convert Binary Number toDecimal and vice-versa

```
fun main(args: Array<String>) {
    val num: Long = 10
    val decimal = convertBinaryToDecimal(num)
    println("$num in binary = $decimal in decimal")
}
fun convertBinaryToDecimal(num: Long): Int {
    var num = num
    var decimalNumber = 0
    var i = 0
    var remainder: Long

    while (num.toInt() != 0) {
        remainder = num % 10
        num /= 10
```

```
        decimalNumber += (remainder * Math.pow(2.0, i.toDouble())).toInt()
        ++i
    }
    return decimalNumber
}
```

B13. Kotlin Program to Reverse a Sentence Using Recursion

```
fun main(args: Array<String>) {
    val san = "Hello"
    println(reverse(san))
}
fun reverse(sentence: String): String {
    if (sentence.isEmpty())
        return sentence
    return reverse(sentence.substring(1)) + sentence[0]
}
```

B14. Kotlin Program to calculate the power using recursion

```
fun main(args: Array<String>) {
    println(pow(2,4))
}
var count = 1
fun pow(number:Int,root:Int):Int{
  if (root<1){
     return count
  }
    else{
     count = count*number
     return pow(number,root-1)
  }
}
```

B15. WAP to print length of String with smart cast. (use “is” operator)

```
fun main(args: Array<String>) {
    println(lenghofstring("pratik"))
}
  fun lenghofstring(string: String){
      if(string is String) println("The length of String is ${string.length}")
       else println("Your entered input is not  String")
  }
```

B16. WAP to check whether the number lies in range. (in operator)

```
fun main(args: Array<String>) {
    println(inrange(60))
}
  fun inrange(a:Int){
      if(a in 1..30) {println("The given number is in range")}
       else { println("Your entered number is not in range")}
  }
```

B17. WAP to check whether the number not lies in range. (!in operator)

```
fun main(args: Array<String>) {
    println(inrange(60))
}
  fun inrange(a:Int){
      if(a !in 1..30) {println("Your entered number is not in range")}
       else {println("The given number is in range") }
  }
```

B18. WAP to demonstrate the use of lazy property. (lazy)

```
fun main(args: Array<String>) {
    println(lazyValue)
    println(lazyValue)
}
val lazyValue: String by lazy {
```

```
    "pratik"
}
```

B19. WAP to demonstrate the use of observable property. (observable)

B20. WAP to demonstrate the use of not null property.(not null with Delegates)

```
fun main(args: Array<String>) {

    var a:String? = "f"
    if (a!!.length > 0){
        print("String is not null")
    }

}
```

B21. WAP to filter array with even numbers only (Reference to function, use :: operator)

B22. WAP to filter array of strings. Print only strings which length not even. Create two function one for length of string second for check not even. Use composition function concept.(composition function)

```
fun main(args: Array<String>) {
    val arr = arrayOf("hello","pratik","abc","avft")
    noeven(arr)
}
fun noeven(arr:Array<String>){
    for (i in arr){
        if (lenghth(i)%2 == 0){
            println(i)
        }
    }
}
fun lenghth(i:String):Int{
```

```
    return i.length
}
```

B23. WAP to implement the sum() function so that it computes the sum of all elements in the given array a. (fun sum(arr:IntArray):Int) (Advanced)

```
fun main(args: Array<String>) {
    val arr = IntArray(6 )
    arr[0] = 1
    arr[1] = 2
    println(sum(arr))
}
fun sum(arr:IntArray):Int{
    var sum = 0
    for(i in arr){
        sum = sum + i
    }
    return sum
}
```

B25. WAP to check given is palindrome or not. (fun isPalindrome(s: String): Boolean)

```
fun main(args: Array<String>) {
 val p = "121"
    pelindrom(p)
}
fun pelindrom(p:String){
    if (p == p.reversed()){
        println("Given number is palindrome ")
    }else println("given string is not palindrome ")
}
```

B26. WAP to convert following 4 function to 1 function using default arguments in kotlin. public String foo(String name, int number, boolean toUpperCase) {

return (toUpperCase ? name.toUpperCase() : name) + number; } public String foo(String name, int number) { return foo(name, number, false); } public String foo(String name, boolean toUpperCase) { return foo(name, 42, toUpperCase); } public String foo(String name) { return foo(name, 42); }

```
fun main(args: Array<String>) {

    println(foo("pratik", toUpperCase = true))

}
fun foo(name: String = "", number: Int = 0, toUpperCase: Boolean = false):
String? {
    return (if (toUpperCase) name.uppercase(Locale.getDefault()) else
name) +
            number
}
```