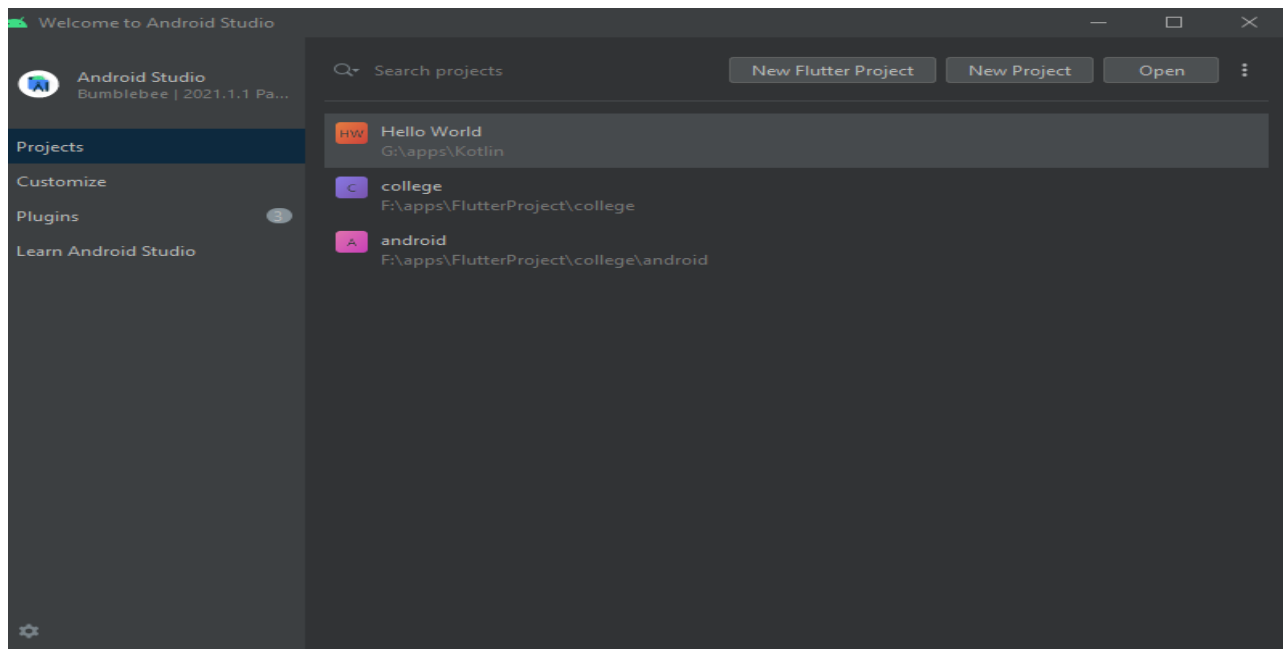


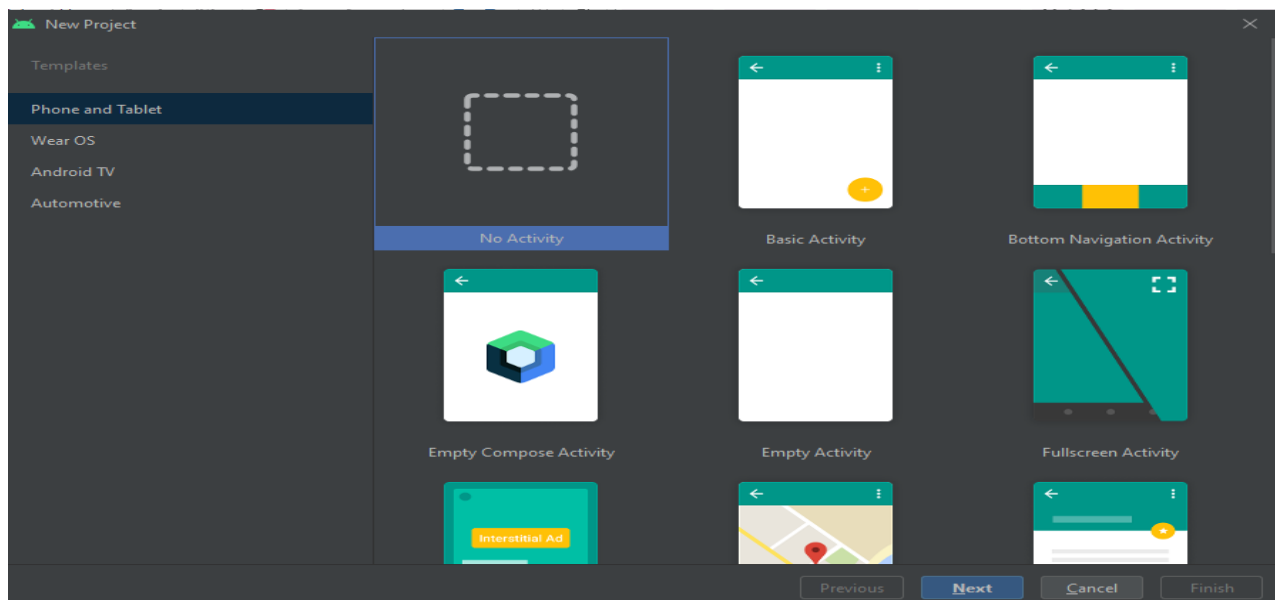
Unit - 1 Introduction to Kotlin Programming

How to Create and Run Kotlin Programme in Android Studio :-

Step - 1 : Open Android Studio and Click on New Project

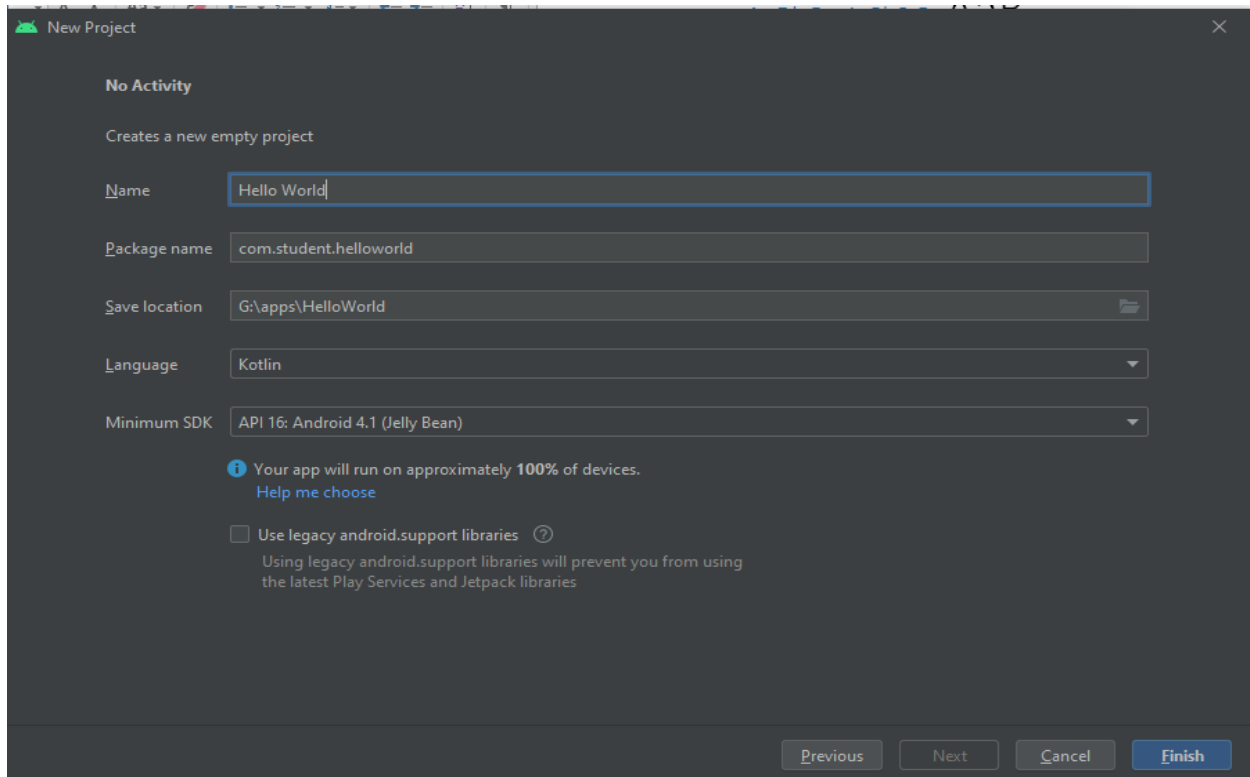


Step - 2 : Select No Activity and Click Next



Step - 3 : Provide the project Details in new Window Like this Fill it And Finish

- ➔ Name : Enter Project Name
- ➔ Package Name : Check Your Package Name (If you want to change you can Change it)
- ➔ Save Location : Where you can save
- ➔ Language : Select Kotlin



New Project

No Activity

Creates a new empty project

Name: Hello World

Package name: com.student.helloworld

Save location: G:\apps\HelloWorld

Language: Kotlin

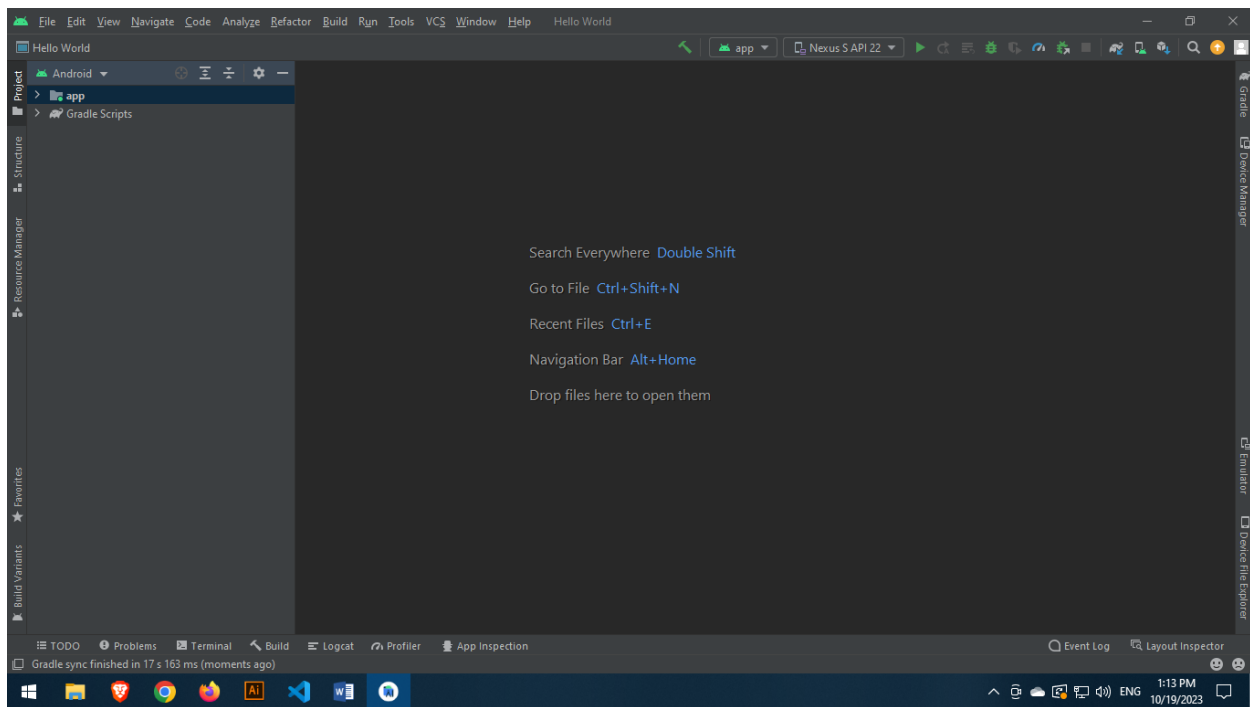
Minimum SDK: API 16: Android 4.1 (Jelly Bean)

i Your app will run on approximately **100%** of devices.
[Help me choose](#)

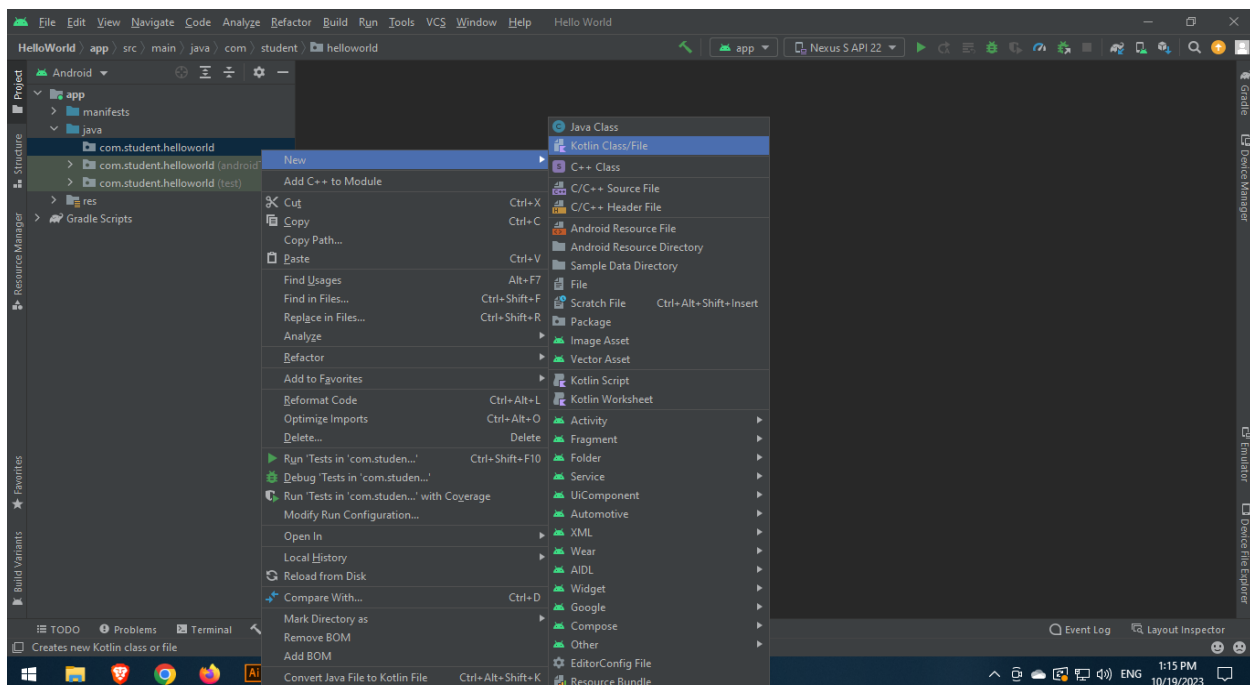
☐ Use legacy android.support libraries **?**
Using legacy android.support libraries will prevent you from using the latest Play Services and Jetpack libraries

Previous Next Cancel Finish

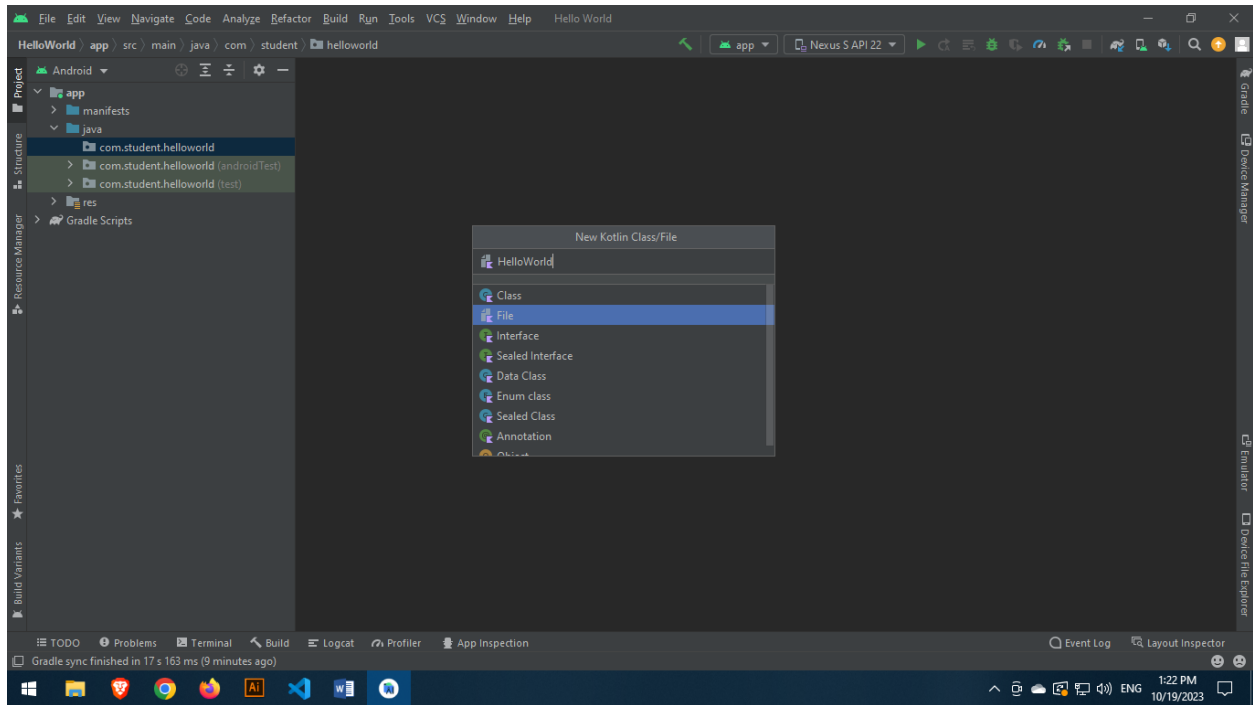
Step - 4 : Then Open This type of Window



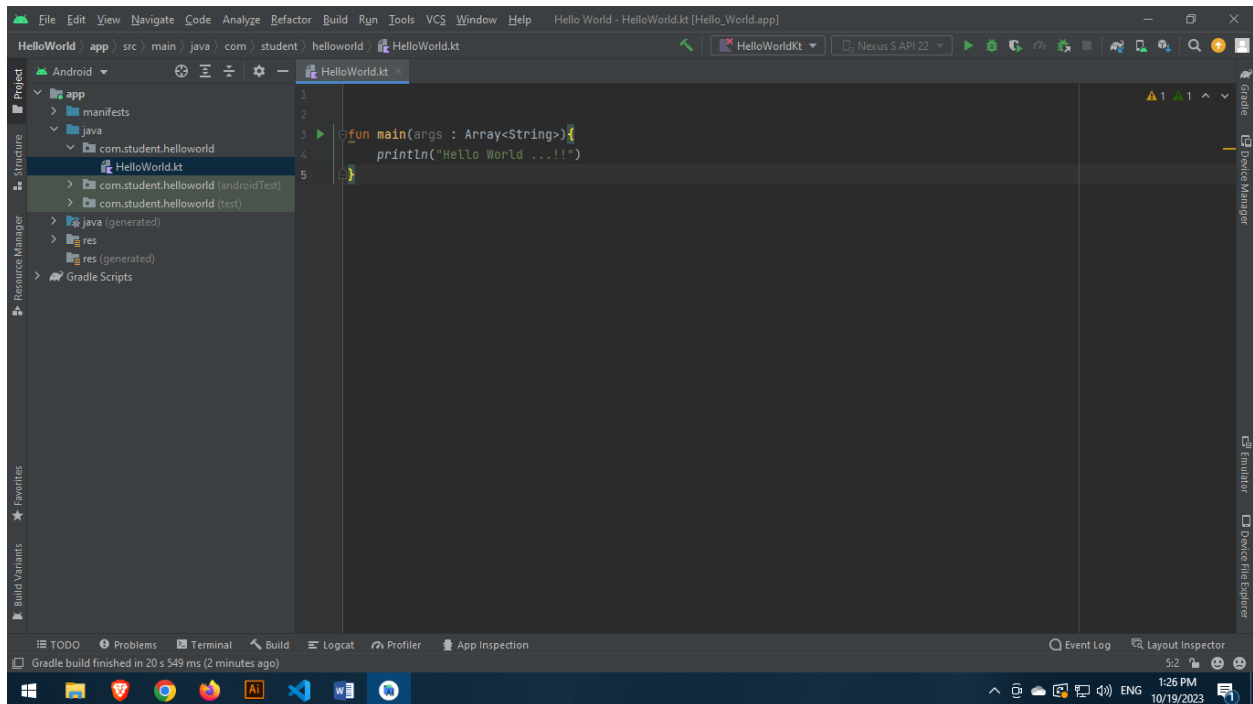
Step - 5 : Create a new Kotlin file -> app -> java -> com.student.helloworld (your package name) -> Select and Right Click onto the package name of file -> new -> kotlin Class/File and then Click .



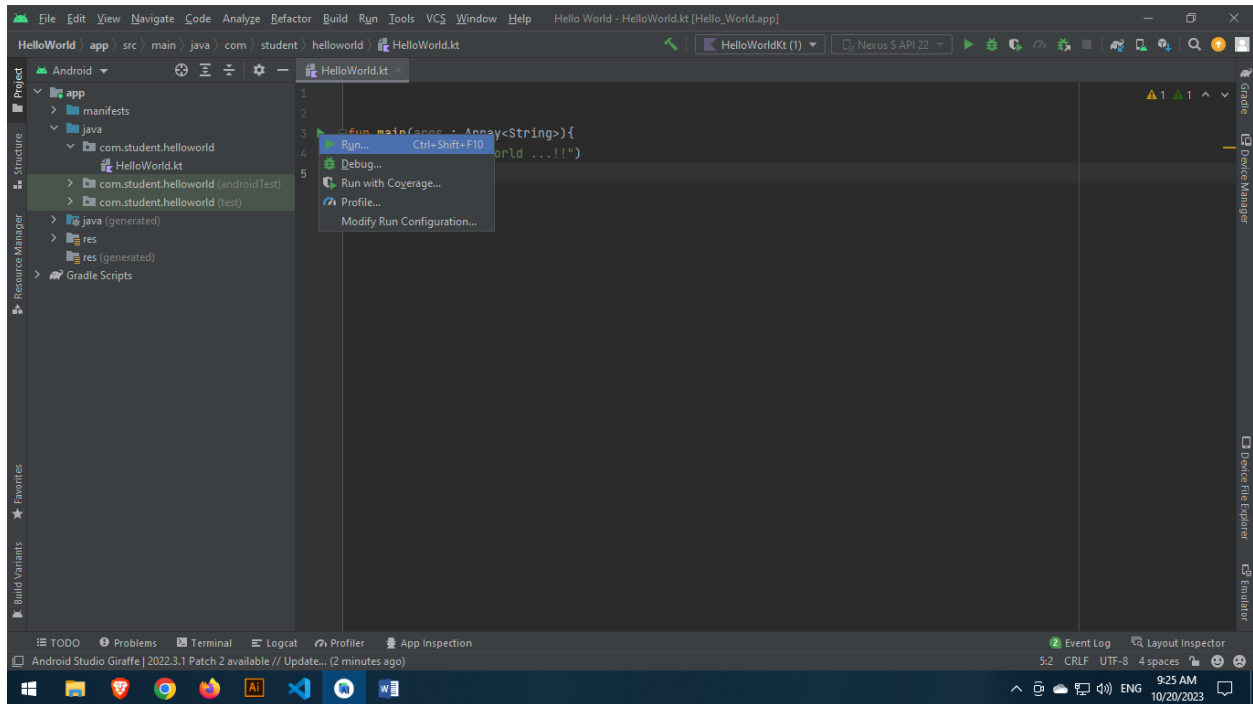
Step - 6 : Enter the File and then press Enter Key



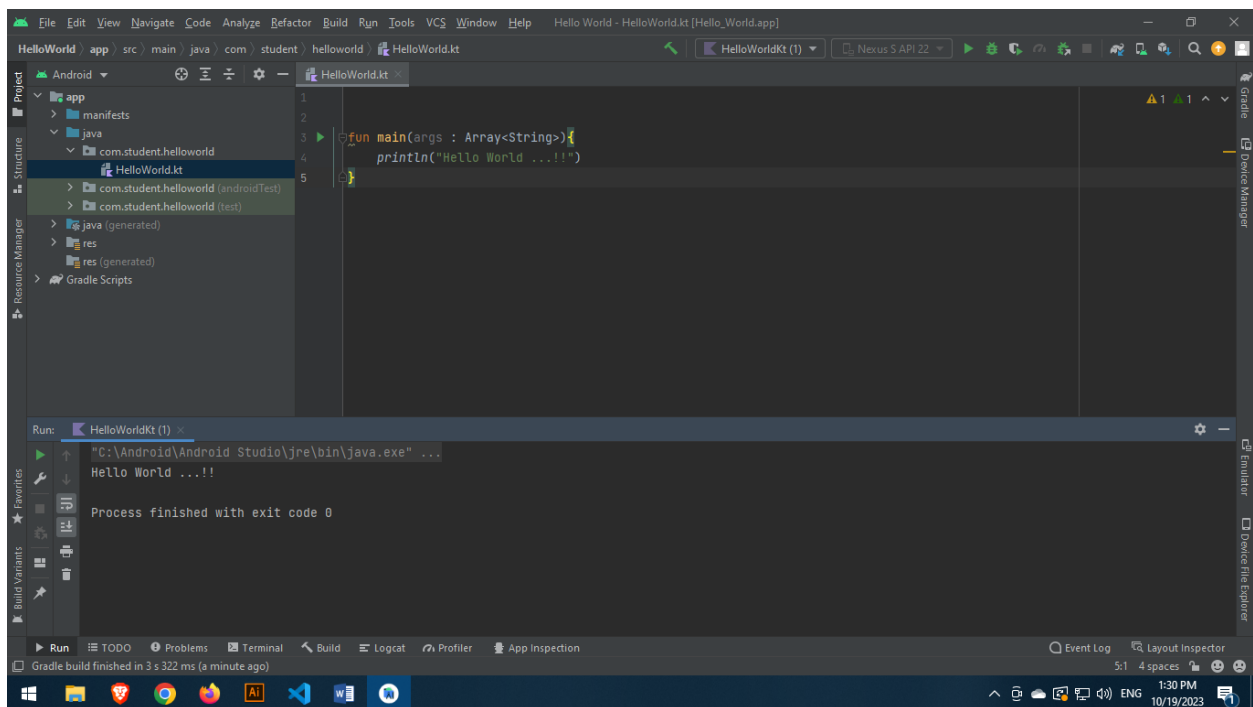
Step - 7 : Remove Package name and Type your kotlin programe Code



Step - 8 : Left side of the fun display to Green play button click on and Select Run (your file name)



And then output is here show in to this bottom of the window



Basics of Kotlin

Progame - 1 : HelloWorld.kt

```
fun main(args : Array<String>){  
    println("Hello World ...!!")  
}
```

Decision Making

Progame - 2 : If Condition.kt

```
fun main(args : Array<String>){  
    println("If Condition")  
    var a = 20  
    if (a<40){  
        println("Number is Smaller then 40")  
    }  
}
```

Progame - 3 : If - else Condition.kt

```
fun main(args : Array<String>){  
    println("If-else Condition")  
    val a = 20  
    val b = 30  
  
    val result = if (a > b){  
        println("$a is grater then $b")  
    }else{  
        println("$a is smaller then $b")  
    }  
    println(result)  
}
```

Progame - 4 : If - else & else - if Condition.kt (Ladder Expression)

```
fun main(args : Array<String>){  
    println("Nested If Expression")  
    val marks = 74
```

```

if (marks > 85){
    println("Excellent")
}else if (marks > 75){
    println("Very Good")
}else if (marks > 65){
    println("Good")
}else{
    println("Average")
}
}

```

Progame - 5 : Nested if Expression

```

fun main(args : Array<String>){
    println("Nested If Expression")
    var a = 10
    var b = 20
    var c = 30

    if (a > b){
        if (a > c){
            println("a is grater - (a>b)")
        }else{
            println("c is grater - (a>c)")
        }
    }else if (b > c){
        println("b is grater - (b>c)")
    }else{
        println("c is grater")
    }
}

```

Loop Control

Progame - 6 : for loop (iterating the elements of array)

```

fun main(args : Array<String>){
    println("For Loop Iterate through array")

    val marks = arrayOf(10,20,30,80,90,100)
    for (item in marks){

```

```

        println(item)
    }
}

```

Progame – 6.1 : for loop (Enclose within curly braces {})

```

fun main(args : Array<String>){
    println("For Loop Iterate through array")

    val marks = arrayOf(10,20,30,80,90,100)
    for (item in marks)
        println(item)
}

```

Progame – 6.2 : for loop (basis of indices (index) of array)

```

fun main(args : Array<String>){
    println("For Loop Iterate through basis of indices (index) of array")

    val marks = arrayOf(10,20,30,80,90,100)
    for (item in marks.indices)
        println("marks[$item] : "+marks[item])
}

```

Progame – 6.3 : for loop (through Range)

```

fun main(args : Array<String>){
    println("For Loop Iterate through range")

    print("for (i in 1..5) print(i) = ")
    for (i in 1..5) print(i)
    println()
    print("for (i in 5..1) print(i) = ")
    for (i in 5..1) print(i)          // prints nothing
    println()
    print("for (i in 5 downTo 1) print(i) = ")
    for (i in 5 downTo 1) print(i)
    println()
    print("for (i in 5 downTo 2) print(i) = ")
    for (i in 5 downTo 2) print(i)
    println()
    print("for (i in 1..5 step 2) print(i) = ")
}

```



```

    for (i in 1..5 step 2) print(i)
    println()
    print("for (i in 5 downTo 1 step 2) print(i) = ")
    for (i in 5 downTo 1 step 2) print(i)
}

```

Progame - 7 : while loop

```

fun main(args : Array<String>){
    println("While loop")

    var i = 1
    while (i <= 10){
        println(i)
        i++
    }
}

```

Progame - 7.1 : while infinite loop

```

fun main(args : Array<String>){
    println("While Infinite loop")

    while (true){
        println("Infinite Loop")
    }
}

```

Progame - 8 : Do while loop

```

fun main(args : Array<String>){
    println("Do While loop")

    var i = 1
    do {
        println(i)
        i++
    }while (i <= 10)
}

```

Progame - 8.1 : Do while loop (Condition of while false)

```
fun main(args : Array<String>){  
    println("Do While loop execute at once time even the condition of while is  
false")
```

```
  
    var i = 6  
    do {  
        println(i)  
        i++  
    }while (i <= 5)  
}
```

Progame - 9 : Brack Expression

```
fun main(args : Array<String>){  
    println("Break Expression")
```

```
  
    for (i in 1..5){  
        if (i==3){  
            break  
        }  
        println(i)  
    }  
}
```

Progame - 9.1 :Labeled Brack Expression

```
fun main(args : Array<String>){  
    println("Labeled Break Expression")
```

```
  
    loop@ for (i in 1..3) {  
        for (j in 1..3) {  
            println("i = $i and j = $j")  
            if (i == 2)  
                break@loop  
        }  
    }  
}
```

Progame - 10 :Continue Expression

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

```
println("Continue Expression")

for (i in 1..3) {
    println("i = $i")
    if (i == 2) {
        continue
    }
    println("this is below if")
}
}
```

Progame – 10.1 : Labeled Continue Expression

```
fun main(args : Array<String>){
    println("Label Continue Expression")

    labelname@ for (i in 1..3) {
        for (j in 1..3) {
            println("i = $i and j = $j")
            if (i == 2) {
                continue@labelname
            }
            println("this is below if")
        }
    }
}
```

Functions

Progame – 11 : Function (sqrt()) is a library function which returns square root of a number (Double value))

//Standard Library Function

```
fun main(args : Array<String>){
    println("Standard Function")

    var number = 25
    var result = Math.sqrt(number.toDouble())
    print("Square root of $number is $result")
}
```

Progame – 11.1 : Function (Simple Function)

//User defined function

```
fun main(args : Array<String>){
    println("Simple Function")

    sum() //function Name
    print("code after sum")
}
fun sum(){
    var num1 =5
    var num2 = 6
    println("sum = "+(num1+num2))
}
```

Progame – 11.2 : Function (Parameterize Function)

```
fun main(args : Array<String>){
    println("Parameterize Function")

    val result = sum(5, 6)
    print(result)
}
fun sum(number1: Int, number2:Int): Int{
    val add = number1+number2
    return add
}
```

Progame – 12 : Recursion Function (Finite times)

```
var count = 0
fun rec(){
    count++;
    if(count<=5){
        println("hello "+count);
        rec();
    }
}
fun main(args: Array<String>) {
    println("Recursion Function")
}
```

```

    rec();
}

```

Progame – 12.1 : Recursion Function (Factorial Number)

```

fun main(args : Array<String>){
    println("Factorial Number")

    val number = 5
    val result: Long
    result = factorial(number)
    println("Factorial of $number = $result")
}

fun factorial(n: Int): Long {
    return if(n == 1){
        n.toLong()
    }
    else{
        n*factorial(n-1)
    }
}

```

Progame – 12.2 :Tail Recursion Function (calculating sun of nth(100000) number)

```

fun main(args : Array<String>){
    println("Calculating Sun of Nth(100000) Number")

    var number = 100000.toLong()
    var result = recursiveSum(number)
    println("sun of upto $number number = $result")
}

tailrec fun recursiveSum(n: Long, semireresult: Long = 0) : Long {
    return if (n <= 0) {
        semireresult
    } else {
        recursiveSum( (n - 1), n+semireresult)
    }
}

```

Progame – 12.3 :Tail Recursion Function (calculating factorial of number)

```
fun main(args : Array<String>){
    println("Calculating factorial of Number")

    val number = 4
    val result: Long
    result = factorial(number)
    println("Factorial of $number = $result")
}

tailrec fun factorial(n: Int, run: Int = 1): Long {
    return if (n == 1){
        run.toLong()
    } else {
        factorial(n-1, run*n)
    }
}
```

Progame – 13 : Default Argument (passing no argument in function call)

```
fun main(args : Array<String>){
    println("Passing no argument in function call")

    run()
}

fun run(num:Int= 5, latter: Char ='x'){
    print("parameter in function definition $num and $latter")
}
```

Progame – 13.1 : Default Argument (passing some argument in function call)

```
fun main(args : Array<String>){
    println("Passing some argument in function call")

    run(3)
```

```

}
fun run(num:Int= 5, latter: Char ='x'){
    print("parameter in function definition $num and $latter")
}

```

Progame - 13.2 : Default Argument (passing all argument in function call)

```

fun main(args : Array<String>){
    println("Passing all argument in function call")

    run(3,'a')
}
fun run(num:Int= 5, latter: Char ='x'){
    print("parameter in function definition $num and $latter")
}

```

Progame - 14 : Named Argument

```

fun main(args : Array<String>){
    println("Named Argument")

    run(latter='a')
}
fun run(num:Int= 5, latter: Char ='x'){
    print("parameter in function definition $num and $latter")
}

```

Progame - 15 : Lambda Function (Normal function: addition of two numbers)

```

fun main(args : Array<String>){
    println("Lambda Function (Normal function)")

    addNumber(5,10)
}
fun addNumber(a: Int, b: Int){
    val add = a + b
    println(add)
}

```

Progame – 15.1 : Lambda Function (Normal function: addition of two numbers)

```
fun main(args : Array<String>){
    println("Lambda Function (Normal function)")

    val myLambda: (Int) -> Unit= {s: Int -> println(s) } //lambda function
    addNumber(5,10,myLambda)
}
fun addNumber(a: Int, b: Int, mylambda: (Int) -> Unit ){ //high level function
    lambda as parameter
    val add = a + b
    mylambda(add) // println(add)
}
```

Progame – 16 : Higher order function

- ➔ High order function (Higher level function) is a function which accepts function as a parameter or returns a function or can do both. Means, instead of passing Int, String, or other types as a parameter in a function we can pass a function as a parameter in other function.

```
fun myFun(org: String,portal: String, fn: (String,String) -> String): Unit {
    val result = fn(org,portal)
    println(result)
}

fun main(args: Array<String>){
    val fn:(String,String)->String={org,portal->"$org develop by $portal"}
    myFun("Youtube.com","Google",fn)
}
```

Progame – 17 : Inline function

```
fun main(args : Array<String>){
    println("Inline Function")

    inlineFunction({ println("calling inline functions")})
}

inline fun inlineFunction(myFun: () -> Unit ) {
    myFun()
}
```



```

    print("code inside inline function")
}

```

Progame – 17.1 : Inline function(Non local control flow)

```

fun main(args : Array<String>){
    inlineFunction({ println("calling inline functions")
        return},{ println("next parameter in inline functions")})
}

```

```

inline fun inlineFunction(myFun: () -> Unit, nxtFun: () -> Unit) {
    myFun()
    nxtFun()
    print("code inside inline function")
}

```

Progame – 17.2 : Inline function(crossinline annotation)

```

fun main(args : Array<String>){
    inlineFunction({ println("calling inline functions")
        return@inlineFunction // compile time error
    },{ println("next parameter in inline functions")})
}

```

```

inline fun inlineFunction(crossinline myFun: () -> Unit, nxtFun: () -> Unit) {
    myFun()
    nxtFun()
    print("code inside inline function")
}

```

Progame – 17.3 : Inline function(noinline modifier)

```

fun main(args : Array<String>){
    inlineFunctionExample({ println("calling inline functions")},
        { println("next parameter in inline functions")})

    println("this is main function closing")
}

```

```

inline fun inlineFunctionExample(myFun: () -> Unit, noinline nxtFun: () -> Unit )
{
    myFun()
}

```

```

    nxtFun()
    println("code inside inline function")
}

```

Data Structures (Collections)

There are a three types of Collection :

- 1) List
- 2) Map
- 3) Set

Progame - 18 : Array

```

fun main(args : Array<String>){
    var myArray = Array<Int>(5){0}

    for(element in myArray){
        println(element)
    }
}

```

Progame - 18.1 : Array (mutable)

```

fun main(args : Array<String>){
    var myArray = Array<Int>(5){0}

    myArray[1]= 10
    myArray[3]= 15

    for(element in myArray){
        println(element)
    }
}

```

Progame - 18.2 : Array (arrayOf() and intArrayOf() function)

```

fun main(args : Array<String>){
    val name = arrayOf<String>("Windows","Linux","MacOS","Android","IOS")
    var myArray2 = arrayOf<Int>(1,10,4,6,15)
    var myArray3 = arrayOf(5,10,20,12,15)
    var myArray4= arrayOf(1,10,4, "Windows","Linux")
    var myArray5: IntArray = intArrayOf(5,10,20,12,15)
}

```

```

for(element in name){
    println(element)
}

println()
for(element in myArray2){
    println(element)
}
println()
for(element in myArray3){
    println(element)
}
println()
for(element in myArray4){
    println(element)
}
println()
for(element in myArray5){
    println(element)
}
}

```

Progame - 18.3 : Array (traversing using range)

```

fun main(args : Array<String>){
    var myArray5: IntArray = intArrayOf(5,10,20,12,15)

    for (index in 0..4){
        println(myArray5[index])
    }
    println()
    for (index in 0..myArray5.size-1){
        println(myArray5[index])
    }
}

```

Progame - 19 : List Interface (listOf() function)

```

fun main(args : Array<String>){
    var list = listOf("Windows","Linux","MacOS")//read only, fix-size
    for(element in list){
        println(element)
    }
}

```

```

    }
}

```

Progame – 19.1 : List Interface (listOf() function using index range)

```

fun main(args : Array<String>){
    var list = listOf(1,2,3,"Windows","Linux","MacOS")//read only, fix-size
    for(element in list){
        println(element)
    }
    println()
    for(index in 0..list.size-1){
        println(list[index])
    }
}

```

Progame – 19.2 : List Interface (listOf<Int>(), listOf<String>(), listOf<Any>())

```

fun main(args : Array<String>){
    var intList: List<Int> = listOf<Int>(1,2,3)
    var stringList: List<String> = listOf<String>("Windows","Linux","MacOS")
    var anyList: List<Any> = listOf<Any>(1,2,3,"Windows","Linux","MacOS")
    println("print int list")
    for(element in intList){
        println(element)
    }
    println()
    println("print string list")
    for(element in stringList){
        println(element)
    }
    println()
    println("print any list")
    for(element in anyList){
        println(element)
    }
}

```

Progame – 19.2 : List Interface (listOf<T>() function)

```
fun main(args : Array<String>){
    var stringList: List<String> =
listOf<String>("Windows","Linux","MacOS","Android","IOS")
    var list: List<String> = listOf<String>("Windows","Linux","MacOS")
    for(element in stringList){
        print(element+" ")
    }
    println()
    println(stringList.get(0))
    println(stringList.indexOf("Linux"))
    println(stringList.lastIndexOf("Linux"))
    println(stringList.size)
    println(stringList.contains("MacOS"))
    println(stringList.containsAll(list))
    println(stringList.subList(2,4))
    println(stringList.isEmpty())
    println(stringList.drop(1))
    println(stringList.dropLast(2))
}
```

Progame – 20 : MutableList (mutableListOf())

```
fun main(args : Array<String>){
    var mutableList = mutableListOf("Windows","Linux","MacOS","Android")

    for(element in mutableList){
        println(element)
    }
    println()
    for(index in 0..mutableList.size-1){
        println(mutableList[index])
    }
}
```

Progame – 20.1 : MutableList (mutableListOf())

```
fun main(args : Array<String>){
    var mutableList1 = mutableListOf("Windows","Linux")
    mutableList1.add("MacOS")
    mutableList1.add("Linux")
}
```

```

var mutableList2 = mutableListOf<String>()
mutableList2.add("Android")
mutableList2.add("ISO")
mutableList2.add("MIUI")

for(element in mutableList1){
    println(element)
}
println()
for(element in mutableList2){
    println(element)
}
}

```

Progame – 20.2 : MutableList (mutableListOf<Int>(),

mutableListOf<String>(), mutableListOf<Any>())

```

fun main(args : Array<String>){
    var mutableListInt: MutableList<Int> = mutableListOf<Int>()
    var mutableListString: MutableList<String> = mutableListOf<String>()
    var mutableListAny: MutableList<Any> = mutableListOf<Any>()

    mutableListInt.add(5)
    mutableListInt.add(7)
    mutableListInt.add(10)
    mutableListInt.add(2,15) //add element 15 at index 2

    mutableListString.add("Java")
    mutableListString.add("Kotlin")
    mutableListString.add("Python")

    mutableListAny.add("Android")
    mutableListAny.add(2)
    mutableListAny.add(5)
    mutableListAny.add("Asp.Net")

    println(".....print Int type.....")
    for(element in mutableListInt){
        println(element)
    }
}

```

```

println()
println(".....print String type.....")
for(element in mutableListString){
    println(element)
}
println()
println(".....print Any type.....")
for(element in mutableListAny){
    println(element)
}
}

```

Progame – 20.3 : MutableList (mutableListOf<T>() function)

```

fun main(args : Array<String>){
    var mutableList = mutableListOf<String>()

    mutableList.add("Java") // index 0
    mutableList.add("Kotlin") // index 1
    mutableList.add("Python") // index 2

    var mutableList2 = mutableListOf<String>("C Language","Data Structure")
    var mutableList3 = mutableListOf<String>("HTML","C++")
    var mutableList4 = mutableListOf<String>("Java","PHP","C Sharp")

    println(".....mutableList.....")
    for(element in mutableList){
        println(element)
    }
    println(".....mutableList[2].....")
    println(mutableList[2])
    mutableList.add(2,"Oracle")
    println(".....mutableList.add(2,\"Oracle\").....")
    for(element in mutableList){
        println(element)
    }
    mutableList.add("C Sharp")
    println(".....mutableList.add(\"C Sharp\").....")
    for(element in mutableList){
        println(element)
    }
}

```

```

mutableList.addAll(1,mutableList3)
println("... mutableList.addAll(1,mutableList3)....")
for(element in mutableList){
    println(element)
}
mutableList.addAll(mutableList2)
println("...mutableList.addAll(mutableList2)....")
for(element in mutableList){
    println(element)
}
mutableList.remove("Kotlin")
println("...mutableList.remove(\"Kotlin\")....")
for(element in mutableList){
    println(element)
}
mutableList.removeAt(2)
println("....mutableList.removeAt(2)....")
for(element in mutableList){
    println(element)
}
mutableList.removeAll(mutableList2)
println(".... mutableList.removeAll(mutableList2)....")
for(element in mutableList){
    println(element)
}

println("....mutableList.set(2,\"Android\")....")
mutableList.set(2,"Android")
for(element in mutableList){
    println(element)
}

println(".... mutableList.retainAll(mutableList4)....")
mutableList.retainAll(mutableList4)
for(element in mutableList){
    println(element)
}
println(".... mutableList2.clear()....")
mutableList2.clear()

```



```

    for(element in mutableList2){
        println(element)
    }
    println(".... mutableList2 after mutableList2.clear()....")
    println(mutableList2)

    println("....mutableList.subList(1,2)....")
    println(mutableList.subList(1,2))
}

```

Progame – 21 : ArrayList (empty ArrayList)

```

fun main(args : Array<String>){
    val arrayList = ArrayList<String>()//Creating an empty arraylist
    arrayList.add("Java")// Adding object in arraylist
    arrayList.add("Kotlin")
    arrayList.add("Python")
    arrayList.add("Oracle")
    arrayList.add("C Sharp")
    println(".....print ArrayList.....")
    for (i in arrayList) {
        println(i)
    }
}

```

Progame – 21.1 : ArrayList (initialize ArrayList capacity)

```

fun main(args : Array<String>){
    val arrayList1 = ArrayList<String>(5)
    arrayList1.add("Java")// Adding object in arraylist
    arrayList1.add("Kotlin")
    arrayList1.add("Python")
    arrayList1.add("Oracle")
    arrayList1.add("C Sharp")
    println(".....print ArrayList1.....")
    for (i in arrayList1) {
        println(i)
    }
    println("size of arrayList1 = "+arrayList1.size)
    val arrayList2 = ArrayList<Int>(5)
    arrayList2.add(14)
    arrayList2.add(20)
}

```

```

    arrayList2.add(80)
    println(".....print ArrayList2.....")
    for (i in arrayList2) {
        println(i)
    }
    println("size of arrayList2 = "+arrayList2.size)
}

```

Progame – 21.2 : ArrayList (filled elements in ArrayList using collection)

```

fun main(args : Array<String>){
    val arrayList: ArrayList<String> = ArrayList<String>(5)
    var list: MutableList<String> = mutableListOf<String>()

    list.add("Java")
    list.add("Kotlin")
    list.add("Python")

    arrayList.addAll(list)
    println(".....print ArrayList.....")
    val itr = arrayList.iterator()
    while(itr.hasNext()) {
        println(itr.next())
    }
    println("size of arrayList = "+arrayList.size)
}

```

Progame – 21.3 : ArrayList (get())

```

fun main(args : Array<String>){
    val arrayList: ArrayList<String> = ArrayList<String>(5)

    arrayList.add("Java")
    arrayList.add("Kotlin")
    arrayList.add("Python")
    arrayList.add("Oracle")
    arrayList.add("C Sharp")
    println(".....print ArrayList.....")
    for (i in arrayList) {
        println(i)
    }
    println(".....arrayList.get(2).....")
}

```

```

    println( arrayList.get(2))
}

```

Progame – 21.4 : ArrayList (set())

```

fun main(args : Array<String>){
    val arrayList: ArrayList<String> = ArrayList<String>(5)

    arrayList.add("Java")
    arrayList.add("Kotlin")
    arrayList.add("Python")
    arrayList.add("Oracle")
    arrayList.add("C Sharp")
    println(".....print ArrayList.....")
    for (i in arrayList) {
        println(i)
    }
    println(".....arrayList.set(2,\"ASP .Net\").....")
    arrayList.set(2,"ASP .Net")
    println(".....print ArrayList.....")
    for (i in arrayList) {
        println(i)
    }
}

```

Progame – 21.5 : ArrayList (indexOf())

```

fun main(args : Array<String>){
    val arrayList: ArrayList<String> = ArrayList<String>(5)

    arrayList.add("Java")
    arrayList.add("Kotlin")
    arrayList.add("Python")
    arrayList.add("Oracle")
    arrayList.add("C Sharp")
    println(".....print ArrayList.....")
    for (i in arrayList) {
        println(i)
    }
    println(".....arrayList.indexOf(\"C Sharp\").....")
    println(arrayList.indexOf("C Sharp"))
}

```

Progame – 21.6 : ArrayList (lastIndexOf())

```
fun main(args : Array<String>){
    val arrayList: ArrayList<String> = ArrayList<String>(5)

    arrayList.add("Java")
    arrayList.add("Kotlin")
    arrayList.add("Python")
    arrayList.add("Oracle")
    arrayList.add("C Sharp")
    println(".....print ArrayList.....")
    for (i in arrayList) {
        println(i)
    }
    println(".....arrayList.lastIndexOf(\"C Sharp\").....")
    println(arrayList.lastIndexOf("C Sharp"))
}
```

Progame – 21.7 : ArrayList (remove())

```
fun main(args : Array<String>){
    val arrayList: ArrayList<String> = ArrayList<String>(5)

    arrayList.add("Java")
    arrayList.add("Kotlin")
    arrayList.add("Python")
    arrayList.add("Oracle")
    arrayList.add("C Sharp")
    println(".....print ArrayList.....")
    for (i in arrayList) {
        println(i)
    }
    println(".....arrayList.remove(\"C Sharp\").....")
    arrayList.remove("C Sharp")
    for (i in arrayList) {
        println(i)
    }
}
```

Progame – 21.8 : ArrayList (removeAt())

```
fun main(args : Array<String>){
    val arrayList: ArrayList<String> = ArrayList<String>(5)

    arrayList.add("Java")
    arrayList.add("Kotlin")
    arrayList.add("Python")
    arrayList.add("Oracle")
    arrayList.add("C Sharp")
    println(".....print ArrayList.....")
    for (i in arrayList) {
        println(i)
    }
    println(".....arrayList.remove(3).....")
    arrayList.removeAt(3)
    for (i in arrayList) {
        println(i)
    }
}
```

Progame – 21.9 : ArrayList (clear())

```
fun main(args : Array<String>){
    val arrayList: ArrayList<String> = ArrayList<String>(5)

    arrayList.add("Java")
    arrayList.add("Kotlin")
    arrayList.add("Python")
    arrayList.add("Oracle")
    arrayList.add("C Sharp")
    println(".....print ArrayList.....")
    for (i in arrayList) {
        println(i)
    }
    println(".....arrayList.clear().....")
    arrayList.clear()

    for (i in arrayList) {
        println(i)
    }
    println(".....arrayList.....")
}
```

```

    println(arrayList)
}

```

Progame - 22 : ArrayList (arrayListOf() function)

```

fun main(args : Array<String>){
    var arrayList = arrayListOf<Int>(4,7,12)
    for(element in arrayList){
        println(element)
    }
}

```

Progame - 22.1 : ArrayList (arrayListOf<Int>(), arrqayListOf<String>(), arrayListOf<Any>() function)

```

fun main(args : Array<String>){
    var intArrayList: ArrayList<Int> = arrayListOf<Int>(1,2,3)
    var stringArrayList: ArrayList<String> =
arrayListOf<String>("Java","Kotlin","Python")
    var anyArrayList: ArrayList<Any> =
arrayListOf<Any>(1,2,3,"Java","Kotlin","Python")
    println("print int ArrayList")
    for(element in intArrayList){
        println(element)
    }
    println()
    println("print string ArrayList")
    for(element in stringArrayList){
        println(element)
    }
    println()
    println("print any ArrayList")
    for(element in anyArrayList){
        println(element)
    }
}

```

Progame - 22.2 : ArrayList (iterator() function)

```

fun main(args : Array<String>){
    val list: ArrayList<String> = arrayListOf<String>()
}

```

```

list.add("Java")
list.add("Kotlin")
list.add("Python")

println(".....print ArrayList.....")
val itr = list.iterator()
while(itr.hasNext()) {
    println(itr.next())
}
}

```

Progame - 22.3 : ArrayList (get() function)

```

fun main(args : Array<String>){
    val list: ArrayList<String> = arrayListOf<String>()

    list.add("Java")
    list.add("Kotlin")
    list.add("Python")
    list.add("Oracle")
    list.add("C Sharp")
    println(".....print list.....")
    for (i in list) {
        println(i)
    }
    println(".....list.get(2).....")
    println( list.get(2))
}

```

Progame - 22.4 : ArrayList (set() function)

```

fun main(args : Array<String>){
    val list: ArrayList<String> = arrayListOf<String>()

    list.add("Java")
    list.add("Kotlin")
    list.add("Python")

    println(".....print list.....")
    for (i in list) {
        println(i)
    }
}

```

```

println(".....arrayList.set(2,\"Oracle\").....")
list.set(2,"Oracle")
println(".....print ArrayList.....")
for (i in list) {
    println(i)
}
}

```

Progame - 22.5 : ArrayList (arrayListOf() function of ArrayList class - add and print Employee data)

```

class Employee(var id: Int, var name: String, var phone: Int, var city: String)

```

```

fun main(args : Array<String>){
    val arrayList: ArrayList<Employee> = arrayListOf<Employee>()
    val e1 = Employee(101, "Ajay", 55555, "Rajkot")
    val e2 = Employee(102, "Rahul", 44443, "Ahmedabad")
    val e3 = Employee(103, "Sanjay", 45422, "Surat")
    arrayList.add(e1)
    arrayList.add(e2)
    arrayList.add(e3)

    for (e in arrayList) {
        println("${e.id} ${e.name} ${e.phone} ${e.city}")
    }
}

```

Progame - 23 : Map Interface

```

fun main(args : Array<String>){
    val myMap = mapOf<Int,String>(1 to "Java", 4 to "Kotlin", 3 to "Python")
    for(key in myMap.keys){
        println(myMap[key])
    }
}

```

Progame - 23.1 : Map Interface(generic)

```

fun main(args : Array<String>){
    val myMap: Map<Int, String> = mapOf<Int,String>(1 to "Java", 4 to "Kotlin", 3
to "Python")

```



```

    for(key in myMap.keys){
        println("Element at key $key = ${myMap.get(key)}")
    }
}

```

Progame – 23.2 : Map Interface(non generic)

```

fun main(args : Array<String>){
    val myMap = mapOf(1 to "Java", 4 to "Kotlin", 3 to "Python","Oracle" to
"oracle", "two" to 2)
    for(key in myMap.keys){
        println("Element at key $key = ${myMap.get(key)}")
    }
}

```

Progame – 23.3 : Map Interface(mapOf().getValue())

```

fun main(args : Array<String>){
    val myMap: Map<Int,String> = mapOf<Int, String>(1 to "Java", 4 to "Kotlin", 3
to "Python")

    for(key in myMap.keys){
        println("Element at key $key = ${myMap.get(key)}")
    }
    println(".....myMap.getValue(4).....")
    println(myMap.getValue(4))
}

```

Progame – 23.4 : Map Interface(mapOf().contains())

```

fun main(args : Array<String>){
    val myMap: Map<Int,String> = mapOf<Int, String>(1 to "Java", 4 to "Kotlin", 3
to "Python")

    for(key in myMap.keys){
        println("Element at key $key = ${myMap.get(key)}")
    }

    println(".....myMap.contains(3).....")
    println( myMap.contains(3))
}

```

Progame – 23.5 : Map Interface(mapOf().containsKey())

```
fun main(args : Array<String>){
    val myMap: Map<Int,String> = mapOf<Int, String>(1 to "Java", 4 to "Kotlin", 3
to "Python")

    for(key in myMap.keys){
        println("Element at key $key = ${myMap.get(key)}")
    }

    println(".....myMap.containsKey(2).....")
    println(myMap.containsKey(2))
}
```

Progame – 23.6 : Map Interface(mapOf().containsValue())

```
fun main(args : Array<String>){
    val myMap: Map<Int,String> = mapOf<Int, String>(1 to "Java", 4 to "Kotlin", 3
to "Python")

    for(key in myMap.keys){
        println("Element at key $key = ${myMap.get(key)}")
    }
    println(".....myMap.containsValue(\"Java\").....")
    println(myMap.containsValue("Java"))
}
```

Progame – 23.7 : Map Interface(mapOf().get())

```
fun main(args : Array<String>){
    val myMap: Map<Int,String> = mapOf<Int, String>(1 to "Java", 4 to "Kotlin", 3
to "Python")

    for(key in myMap.keys){
        println("Element at key $key = ${myMap.get(key)}")
    }
    println(".....myMap.get(1).....")
    println(myMap.get(1))
}
```

Progame – 23.8 : Map Interface(mapOf().getOrDefault())

```
fun main(args : Array<String>){
    val myMap: Map<Int,String> = mapOf<Int, String>(1 to "Java", 4 to "Kotlin", 3
to "Python")

    for(key in myMap.keys){
        println("Element at key $key = ${myMap.get(key)}")
    }

    println(".....myMap.getOrDefault(3, \"Kotlin\")......")
    println(myMap.getOrDefault(3, "Kotlin"))
}
```

[Note : import packages - import android.os.Build
import androidx.annotation.RequiresApi
And :- @RequiresApi(Build.VERSION_CODES.N)]

Progame – 23.9 : Map Interface(mapOf().asIterable())

```
fun main(args : Array<String>){
    val myMap: Map<Int,String> = mapOf<Int, String>(1 to "Java", 4 to "Kotlin", 3
to "Python")

    for(key in myMap.keys){
        println("Element at key $key = ${myMap.get(key)}")
    }
    println(".....myMap.asIterable()......")
    for(itr in myMap.asIterable()){
        println("key = ${itr.key} value = ${itr.value}")
    }
}
```

Progame – 23.10 : Map Interface(mapOf().iterator())

```
fun main(args : Array<String>){
    val myMap: Map<Int,String> = mapOf<Int, String>(1 to "Java", 4 to "Kotlin", 3
to "Python")

    for(key in myMap.keys){
        println("Element at key $key = ${myMap.get(key)}")
    }
    println(".....myMap.iterator()......")
}
```

```

    for(itr1 in myMap.iterator()){
        println("key = ${itr1.key} value = ${itr1.value}")
    }
}

```

Progame – 23.11 : Map Interface(mapOf().minus())

```

fun main(args : Array<String>){
    val myMap: Map<Int,String> = mapOf<Int, String>(1 to "Java", 4 to "Kotlin", 3
to "Python")

```

```

    for(key in myMap.keys){
        println("Element at key $key = ${myMap.get(key)}")
    }
    println(".....myMap.minus(4).....")
    for(m in myMap.minus(4)){
        println(myMap[m.key])
    }
}

```

Progame – 23.12 : Map Interface(mapOf().plus())

```

fun main(args : Array<String>){
    val myMap: Map<Int,String> = mapOf<Int, String>(1 to "Java", 4 to "Kotlin", 3
to "Python")

```

```

    for(key in myMap.keys){
        println("Element at key $key = ${myMap.get(key)}")
    }
    println(".....myMap.plus(Pair(5, \"Oracle\")).....")
    for(p in myMap.plus(Pair(5, "Oracle"))){
        println("Element at key ${p.key} = ${p.value}")
    }
}

```

Progame – 24 : HashMap(empty)

```

fun main(args : Array<String>){
    val hashMap:HashMap<Int,String> = HashMap<Int,String>() // define empty
hashmap
    hashMap.put(1,"Java")
    hashMap.put(3,"Kotlin")

```

```

hashMap.put(4,"Python")
hashMap.put(2,"Java")
println(".....traversing hashmap.....")
for(key in hashMap.keys){
    println("Element at key $key = ${hashMap[key]}")
}
}

```

Progame – 24.1 : HashMap(initial capacity)

```

fun main(args : Array<String>){
    val hashMap:HashMap<String,String> = HashMap<String,String>(3)
    hashMap.put("name","Unknown")
    hashMap.put("city","Rajkot")
    hashMap.put("department","Software Development")
    println(".....traversing hashmap.....")
    for(key in hashMap.keys){
        println("Element at key $key = ${hashMap[key]}")
    }
    println(".....hashMap.size.....")
    println(hashMap.size)
    hashMap.put("hobby","Travelling")
    println(".....hashMap.size after adding hobby.....")
    println(hashMap.size)
    println(".....traversing hashmap.....")
    for(key in hashMap.keys){
        println("Element at key $key = ${hashMap.get(key)}")
    }
}

```

Progame – 24.2 : HashMap(remove() and put())

```

fun main(args : Array<String>){
    val hashMap:HashMap<Int,String> = HashMap<Int,String>()
    hashMap.put(1,"Java")
    hashMap.put(3,"Kotlin")
    hashMap.put(4,"Python")
    hashMap.put(2,"Oracle")

    println(".....traversing hashmap.....")
    for(key in hashMap.keys){
        println("Element at key $key = ${hashMap[key]}")
    }
}

```

```

    }

    hashMap.replace(3,"C Sharp")
    hashMap.put(2,"Asp .Net")
    println(".....hashMap.replace(3,\"C Sharp\")... hashMap.replace(2,\"Asp
.Net\").....")
    for(key in hashMap.keys){
        println("Element at key $key = ${hashMap[key]}")
    }
}

```

Progame – 24.3 : HashMap(containsKey(key) and containsValue(value))

```

fun main(args : Array<String>){
    val hashMap:HashMap<Int,String> = HashMap<Int,String>()
    hashMap.put(1,"Java")
    hashMap.put(3,"Kotlin")
    hashMap.put(4,"Python")
    hashMap.put(2,"Oracle")

    println(".....traversing hashmap.....")
    for(key in hashMap.keys){
        println("Element at key $key = ${hashMap[key]}")
    }

    println(".....hashMap.containsKey(3).....")
    println(hashMap.containsKey(3))
    println(".....hashMap.containsValue(\"Oracle\").....")
    println(hashMap.containsValue("Oracle"))
}

```

Progame – 24.4 : HashMap(clear())

```

fun main(args : Array<String>){
    val hashMap:HashMap<Int,String> = HashMap<Int,String>()
    hashMap.put(1,"Java")
    hashMap.put(3,"Kotlin")
    hashMap.put(4,"Python")
    hashMap.put(2,"Oracle")

    println(".....traversing hashmap.....")
    for(key in hashMap.keys){

```

```

        println("Element at key $key = ${hashMap[key]}")
    }

    println(".....hashMap.clear().....")
    hashMap.clear()
    println(".....print hashMap after clear().....")
    println(hashMap)
}

```

Progame – 25 : hashMapOf() (hashMapOf<Int, String>(),

hashMapOf<String, String>(), hashMapOf<Any, Any>())

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

```
    val intMap: HashMap<Int, String> = hashMapOf<Int,String>(1 to "Unknown
1",4 to "Unknown 2", 2 to "Unknown 3", 3 to "Unknown 4")
```

```
    val stringMap: HashMap<String,String> = hashMapOf<String,String>("name"
to "Unknown 1")
```

```
    stringMap.put("city", "Rajkot")
```

```
    stringMap.put("department", "Software Development")
```

```
    stringMap.put("hobby", "Playing")
```

```
    val anyMap: HashMap<Any, Any> = hashMapOf<Any, Any>(1 to "Unknown
1", "name" to "Unknown 4", 2 to 200)
```

```
    println(".....traverse intMap.....")
```

```
    for(key in intMap.keys){
```

```
        println(intMap[key])
```

```
    }
```

```
    println(".....traverse stringMap.....")
```

```
    for(key in stringMap.keys){
```

```
        println(stringMap[key])
```

```
    }
```

```
    println(".....traverse anyMap.....")
```

```
    for(key in anyMap.keys){
```

```
        println(anyMap[key])
```

```
    }
```

```
}
```

Progame – 25.1 : hashMapOf() (containsKey())

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

    val stringMap: HashMap<String,String> = hashMapOf<String,String>()
    stringMap.put("name", "Steve")
    stringMap.put("city", "Rajkot")
    stringMap.put("department", "Software Development")
    stringMap.put("hobby", "Playing")
    println(".....traverse stringMap.....")
    for(key in stringMap.keys){
        println("Key = ${key} , value = ${stringMap[key]}")
    }

    println(".....stringMap.containsKey(\"name\").....")
    println(stringMap.containsKey("name"))
}
```

Progame – 25.2 : hashMapOf() (containsValue())

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

    val stringMap: HashMap<String,String> = hashMapOf<String,String>()
    stringMap.put("name", "Steve")
    stringMap.put("city", "Rajkot")
    stringMap.put("department", "Software Development")
    stringMap.put("hobby", "Playing")

    println(".....traverse stringMap.....")
    for(key in stringMap.keys){
        println("Key = ${key} , value = ${stringMap[key]}")
    }

    println(".....stringMap.containsValue(\"Rajkot\").....")
    println(stringMap.containsValue("Rajkot"))
    println(stringMap.containsValue("Ahmedabad"))
}
```


Progame – 25.3 : hashMapOf() (contains())

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

    val stringMap: HashMap<String,String> = hashMapOf<String,String>()
    stringMap.put("name", "Steve")
    stringMap.put("city", "Rajkot")
    stringMap.put("department", "Software Development")
    stringMap.put("hobby", "Playing")

    println(".....traverse stringMap.....")
    for(key in stringMap.keys){
        println("Key = ${key} , value = ${stringMap[key]}")
    }

    println(".....stringMap.contains(\"city\").....")
    println(stringMap.contains("city"))
}
```

Progame – 25.4 : hashMapOf() (replace(key, value))

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

    val stringMap: HashMap<String,String> = hashMapOf<String,String>()
    stringMap.put("name", "Steve")
    stringMap.put("city", "Rajkot")
    stringMap.put("department", "Software Development")
    stringMap.put("hobby", "Playing")

    println(".....traverse stringMap.....")
    for(key in stringMap.keys){
        println("Key = ${key} , value = ${stringMap[key]}")
    }

    println(".....stringMap.replace(\"city\", \"Ahmedabad\").....")
    println(stringMap.replace("city", "Ahmedabad"))
    println(".....traverse stringMap after")
    stringMap.replace(\"city\", \"Ahmedabad\")
    for(key in stringMap.keys){
        println("Key = ${key} , value = ${stringMap[key]}")
    }
}
```

```
}
```

Progame – 25.5 : hashMapOf() (replace(key, oldValue, newValue))

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

    val stringMap: HashMap<String,String> = hashMapOf<String,String>()
    stringMap.put("name", "Steve")
    stringMap.put("city", "Rajkot")
    stringMap.put("department", "Development")
    stringMap.put("hobby", "Playing")

    println(".....traverse stringMap.....")
    for(key in stringMap.keys){
        println("Key = ${key} , value = ${stringMap[key]}")
    }

    println(".....stringMap.replace(\"department\",
    \"Development\", \"Management\").....")
    println(stringMap.replace("department", "Development", "Management"))
    println(".....traverse stringMap after stringMap.replace(\"department\",
    \"Development\", \"Management\").....")
    for(key in stringMap.keys){
        println("Key = ${key} , value = ${stringMap[key]}")
    }
}
```

Progame – 25.6 : hashMapOf() (hashMapOf().size, hashMapOf().key)

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

    val stringMap: HashMap<String,String> = hashMapOf<String,String>()
    stringMap.put("name", "Steve")
    stringMap.put("city", "Rajkot")
    stringMap.put("department", "Development")
    stringMap.put("hobby", "Playing")

    println(".....traverse stringMap.....")
    for(key in stringMap.keys){
        println("Key = ${key} , value = ${stringMap[key]}")
    }
}
```

```

println(".....stringMap.size.....")
println(stringMap.size)

println(".....stringMap.keys.....")
println(stringMap.keys)
}

```

Progame – 25.7 : hashMapOf() (getValue(key), getOrDefault(key, defaultValue))

```

fun main(args : Array<String>){

    val stringMap: HashMap<String,String> = hashMapOf<String,String>()
    stringMap.put("name", "Steve")
    stringMap.put("city", "Rajkot")
    stringMap.put("department", "Development")
    stringMap.put("hobby", "Playing")

    println(".....traverse stringMap.....")
    for(key in stringMap.keys){
        println("Key = ${key} , value = ${stringMap[key]}")
    }

    println(".....stringMap.getValue(\"department\").....")
    println(stringMap.getValue("department"))

    println(".....stringMap.getOrDefault(\"name\", \"Default Value\").....")
    println(stringMap.getOrDefault("name", "Default Value"))
}

```

Progame – 25.8 : hashMapOf() (remove(key))

```

fun main(args : Array<String>){

    val stringMap: HashMap<String,String> = hashMapOf<String,String>()
    stringMap.put("name", "Steve")
    stringMap.put("city", "Rajkot")
    stringMap.put("department", "Development")
    stringMap.put("hobby", "Playing")
}

```

```

println(".....traverse stringMap.....")
for(key in stringMap.keys){
    println("Key = ${key} , value = ${stringMap[key]}")
}

println(".....stringMap.remove(\"city\").....")
println(stringMap.remove("city"))
println(".....traverse stringMap after stringMap.remove(\"city\").....")
for(key in stringMap.keys){
    println("Key = ${key} , value = ${stringMap[key]}")
}
}

```

Progame – 25.9 : hashMapOf() (remove(key , value))

```

fun main(args : Array<String>){

    val stringMap: HashMap<String,String> = hashMapOf<String,String>()
    stringMap.put("name", "Steve")
    stringMap.put("city", "Rajkot")
    stringMap.put("department", "Development")
    stringMap.put("hobby", "Playing")

    println(".....traverse stringMap.....")
    for(key in stringMap.keys){
        println("Key = ${key} , value = ${stringMap[key]}")
    }
    println(".....stringMap.remove(\"hobby\", \"Playing\").....")
    println(stringMap.remove("hobby", "Playing"))
    println(".....traverse stringMap after
stringMap.remove(\"hobby\", \"Playing\").....")
    for(key in stringMap.keys){
        println("Key = ${key} , value = ${stringMap[key]}")
    }
}

```

Progame – 25.10 : hashMapOf() (set(key, value))

```

fun main(args : Array<String>){

    val stringMap: HashMap<String,String> = hashMapOf<String,String>()
    stringMap.put("name", "Steve")

```

```

stringMap.put("city", "Rajkot")
stringMap.put("department", "Development")
stringMap.put("hobby", "Playing")

println(".....traverse stringMap.....")
for(key in stringMap.keys){
    println("Key = ${key} , value = ${stringMap[key]}")
}

stringMap.set("name","User1")
stringMap.set("skill","Kotlin")
println(".....traverse stringMap after stringMap.set(\"name\", \"User1\") and
stringMap.set(\"skill\", \"Kotlin\").....")
for(key in stringMap.keys){
    println("Key = ${key} , value = ${stringMap[key]}")
}
}

```

Progame – 25.11 : hashMapOf() (clear())

```

fun main(args : Array<String>){

    val stringMap: HashMap<String,String> = hashMapOf<String,String>()
    stringMap.put("name", "Steve")
    stringMap.put("city", "Rajkot")
    stringMap.put("department", "Development")
    stringMap.put("hobby", "Playing")

    println(".....traverse stringMap.....")
    for(key in stringMap.keys){
        println("Key = ${key} , value = ${stringMap[key]}")
    }

    println(".....stringMap.clear().....")
    println(stringMap.clear())
    println(stringMap)
}

```

Progame – 26 : Set Interface (setOf())

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

    val intSet = setOf(2,6,4,29,4,5)
    val mySet: Set<Any> = setOf(2,6,4,29,4,5,"Java","Kotlin")
    println(".....print Int set.....")
    for(element in intSet){
        println(element)
    }
    println(".....print Any set.....")
    for(element in mySet){
        println(element)
    }
}
```

Progame – 26.1 : Set Interface (contains() and containsAll())

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

    val mySet: Set<Any> = setOf(2,6,4,29,5,"Java","Kotlin")
    val intSet = setOf(6,4,29)
    println(".....print Any set.....")
    for(element in mySet){
        println(element)
    }
    println("...mySet.contains\ "Java\ "...")
    println(mySet.contains("Java"))
    println("...mySet.contains(20)...")
    println(mySet.contains(20))
    println("...mySet.containsAll(intSet)...")
    println(mySet.containsAll(intSet))
}
```

Progame – 26.2 : Set Interface (isEmpty() and isEmpty())

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

    val mySet: Set<Any> = setOf(2,6,4,29,5,"Java","Kotlin")
    println(".....print Any set.....")
    for(element in mySet){
        println(element)
    }
}
```

```

    }

    println("...mySet.isEmpty()...")
    println(mySet.isEmpty())

    println("...mySet.isNotEmpty()...")
    println(mySet.isNotEmpty())
}

```

Progame – 26.3 : Set Interface (drop())

```

fun main(args : Array<String>){

    val mySet: Set<Any> = setOf(2,6,4,29,4,5,"Java","Kotlin","Java")
    println(".....print Any set.....")
    for(element in mySet){
        println(element)
    }
    val remainList= mySet.drop(4)
    println(".....print Set after mySet.drop(4).....")
    for(element in remainList){
        println(element)
    }
}

```

Progame – 26.4 : Set Interface (elementAt() and elementAtOrNull())

```

fun main(args : Array<String>){

    val mySet: Set<Any> = setOf(2,6,4,29,4,5,"Java","Kotlin","Java")
    println(".....print Any set.....")
    for(element in mySet){
        println(element)
    }

    println(".....print mySet.elementAt(3).....")
    println(mySet.elementAt(3))

    println(".....print mySet.elementAtOrNull(5).....")
    println(mySet.elementAtOrNull(5))
}

```

Object Oriented Programming

Inheritance

Progame - 27 : Inheriting fields from a class

```
open class Base{
    val x = 10
}
class Derived: Base() {
    fun foo() {
        println("x is equal to " + x)
    }
}
fun main(args: Array<String>) {
    val derived = Derived()
    derived.foo()
}
```

Progame - 27.1 : Inheriting methods from a class

```
open class Bird {
    fun fly() {
        println("flying...")
    }
}
class Duck: Bird() {
    fun swim() {
        println("swimming...")
    }
}
fun main(args: Array<String>) {
    val duck = Duck()
    duck.fly()
    duck.swim()
}
```


Progame – 27.2 : Inheritance

```
open class Employee(name: String, age: Int, salary: Float) {
    init {
        println("Name is $name.")
        println("Age is $age")
        println("Salary is $salary")
    }
}

class Programmer(name: String, age: Int, salary:
Float):Employee(name,age,salary){
    fun doProgram() {
        println("programming is my passion.")
    }
}

class Salesman(name: String, age: Int, salary: Float):Employee(name,age,salary){
    fun fieldWork() {
        println("travelling is my hobby.")
    }
}

fun main(args: Array<String>){
    val obj1 = Programmer("Employee1", 25, 40000f)
    obj1.doProgram()
    val obj2 = Salesman("Employee2", 24, 30000f)
    obj2.fieldWork()
}
```

Progame – 27.3 : Inheritance and primary constructor

```
open class Employee(name: String,salary: Float) {
    init {
        println("Name is $name.")
        println("Salary is $salary")
    }
}

class Programmer(name: String, dept: String, salary:
Float):Employee(name,salary){
    init {
        println("Name $name of department $dept with salary $salary.")
    }
    fun doProgram() {
        println("Programming is my passion.")
    }
}
```

```

    }
}
class Salesman(name: String, dept: String, salary: Float):Employee(name,salary){
    init {
        println("Name $name of department $dept with salary $salary.")
    }
    fun fieldWork() {
        println("Travelling is my hobby.")
    }
}
fun main(args: Array<String>){
    val obj1 = Programmer("Employee1", "Development", 40000f)
    obj1.doProgram()
    println()
    val obj2 = Salesman("Employee2", "Marketing", 30000f)
    obj2.fieldWork()
}

```

Progame – 27.4 : Inheritance and secondary constructor

```

open class Patent {

    constructor(name: String, id: Int) {
        println("execute super constructor $name: $id")
    }
}

class Child: Patent {

    constructor(name: String, id: Int, dept: String): super(name, id) {
        print("execute child class constructor with property $name, $id, $dept")
    }
}
fun main(args: Array<String>) {
    val child = Child("Employee",101, "Developer")
}

```

Progame – 27.5 : Method Overriding (Inheritance without overriding)

```
open class Bird {
    open fun fly() {
        println("Bird is flying...")
    }
}
class Parrot: Bird() {

}
class Duck: Bird() {

}
fun main(args: Array<String>) {
    val p = Parrot()
    p.fly()
    val d = Duck()
    d.fly()
}
```

Progame – 27.6 : Method Overriding (Inheritance method overriding - fly())

```
open class Bird {
    open fun fly() {
        println("Bird is flying...")
    }
}
class Parrot: Bird() {
    override fun fly() {
        println("Parrot is flying...")
    }
}
class Duck: Bird() {
    override fun fly() {
        println("Duck is flying...")
    }
}
fun main(args: Array<String>) {
    val p = Parrot()
```

```

    p.fly()
    val d = Duck()
    d.fly()
}

```

Progame - 27.7 : Method Overriding (Inheritance property overriding)

```

open class Bird {
    open var color = "Black"
    open fun fly() {
        println("Bird is flying...")
    }
}

class Parrot: Bird() {
    override var color = "Green"
    override fun fly() {
        println("Parrot is flying...")
    }
}

class Duck: Bird() {
    override var color = "White"
    override fun fly() {
        println("Duck is flying...")
    }
}

fun main(args: Array<String>) {
    val p = Parrot()
    p.fly()
    println(p.color)
    val d = Duck()
    d.fly()
    println(d.color)
}

```

Progame – 27.8 : Method Overriding (superclass implementation)

```
open class Bird {
    open var color = "Black"
    open fun fly() {
        println("Bird is flying...")
    }
}

class Parrot: Bird() {
    override var color = "Green"
    override fun fly() {
        super.fly()
        println("Parrot is flying...")
    }
}

fun main(args: Array<String>) {
    val p = Parrot()
    p.fly()
    println(p.color)
}
```

Progame – 27.9 : Method Overriding (multiple class implementation)

```
open class Bird {
    open var color = "Black"
    open fun fly() {
        println("Bird is flying...")
    }
}

interface Duck {
    fun fly() {
        println("Duck is flying...")
    }
}

class Parrot: Bird(), Duck {
    override var color = "Green"
    override fun fly() {
        super<Bird>.fly()
        super<Duck>.fly()
    }
}
```

```

        println("Parrot is flying...")
    }
}
fun main(args: Array<String>) {
    val p = Parrot()
    p.fly()
    println(p.color)
}

```

Abstract

Progame - 28 :

```

abstract class Car{
    abstract fun run()
}
class Honda: Car(){
    override fun run(){
        println("Honda is running safely..")
    }
}
fun main(args: Array<String>){
    val obj = Honda()
    obj.run();
}

```

Progame - 28.1 :

```

open class Car {
    open fun run() {
        println("Car is running..")
    }
}
abstract class Honda : Car() {
    override abstract fun run()
}
class City: Honda(){
    override fun run() {

```

```

        println("Honda City is running..")
    }
}
fun main(args: Array<String>){
    val car = Car()
    car.run()
    val city = City()
    city.run()
}

```

Progame – 28.2 : real scenario of abstract class

```

abstract class Bank {
    abstract fun simpleInterest(p: Int, r: Double, t: Int) :Double
}

class SBI : Bank() {
    override fun simpleInterest(p: Int, r: Double, t: Int): Double{
        return (p*r*t)/100
    }
}

class PNB : Bank() {
    override fun simpleInterest(p: Int, r: Double, t: Int): Double{
        return (p*r*t)/100
    }
}

fun main(args: Array<String>) {
    var sbi: Bank = SBI()
    val sbiint = sbi.simpleInterest(1000,5.0,3)
    println("SBI interest is $sbiint")
    var pnb: Bank = PNB()
    val pnbint = pnb.simpleInterest(1000,4.5,3)
    println("PNB interest is $pnbint")
}

```

Interface

Progame – 29 : Implementing Interfaces

```
interface MyInterface {
    var id: Int          // abstract property
    fun absMethod():String // abstract method
    fun doSomething() {
        println("MyInterface doing some work")
    }
}

class InterfaceImp : MyInterface {
    override var id: Int = 101
    override fun absMethod(): String{
        return "Implementing abstract method.."
    }
}

fun main(args: Array<String>) {
    val obj = InterfaceImp()
    println("Calling overriding id value = ${obj.id}")
    obj.doSomething()
    println(obj.absMethod())
}
```

Progame – 29.1 : Implementing multiple interface

```
interface MyInterface1 {
    fun doSomething()
}

interface MyInterface2 {
    fun absMethod()
}

class MyClass : MyInterface1, MyInterface2 {
    override fun doSomething() {
        println("overriding doSomething() of MyInterface1")
    }

    override fun absMethod() {
        println("overriding absMethod() of MyInterface2")
    }
}
```



```

fun main(args: Array<String>) {
    val myClass = MyClass()
    myClass.doSomething()
    myClass.absMethod()
}

```

Progame – 29.2 : method overriding using super keyword as
super<interface_name>.methodName()

```

interface MyInterface1 {
    fun doSomething() {
        println("MyInterface 1 doing some work")
    }
    fun absMethod()
}
interface MyInterface2 {
    fun doSomething(){
        println("MyInterface 2 doing some work")
    }
    fun absMethod(name: String)
}
class MyClass : MyInterface1, MyInterface2 {
    override fun doSomething() {
        super<MyInterface2>.doSomething()
    }

    override fun absMethod() {
        println("Implements absMethod() of MyInterface1")
    }
    override fun absMethod(n: String) {
        println("Implements absMethod(name) of MyInterface2 name is $n")
    }
}
fun main(args: Array<String>) {
    val myClass = MyClass()
    myClass.doSomething()
    myClass.absMethod()
    myClass.absMethod("Unknown")
}

```

Progame – 29.2.1 : method overriding using super keyword as

```
super<interface_name>.methodName()
interface MyInterface1 {
    fun doSomething() {
        println("MyInterface 1 doing some work")
    }
    fun absMethod()
}
interface MyInterface2 {
    fun doSomething() {
        println("MyInterface 2 doing some work")
    }
    fun absMethod() {
        println("MyInterface 2 absMethod")
    }
}

class C : MyInterface1 {
    override fun absMethod() {
        println("MyInterface1 absMethod implementation")
    }
}

class D : MyInterface1, MyInterface2 {
    override fun doSomething() {
        super<MyInterface1>.doSomething()
        super<MyInterface2>.doSomething()
    }
    override fun absMethod() {
        super<MyInterface2>.absMethod()
    }
}

fun main(args: Array<String>) {
    val d = D()
    val c = C()
    d.doSomething()
    d.absMethod()
    c.doSomething()
    c.absMethod()
}
```

1) HelloWorld

activity_main.xml file - (For Designing)

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</android.support.constraint.ConstraintLayout>
```

MainActivity.kt file (Kotlin File)

```
package com.student.helloworld

import android.support.v7.app.AppCompatActivity
import android.os.Bundle

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
}
```

2) Toast Message

activity_main.xml file

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:text="Click to display toast"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</android.support.constraint.ConstraintLayout>
```

MainActivity.kt file

```
package com.student.customtoast

import android.support.v7.app.AppCompatActivity
import android.os.Bundle
import android.view.Gravity
import android.widget.Button
import android.widget.Toast

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        var button = findViewById<Button>(R.id.button)

        button.setOnClickListener() {

            //Short length of Display Message with Toast
            Toast.makeText(applicationContext,"this is toast
message",Toast.LENGTH_SHORT).show()
        }
    }
}
```

```

        //Long length of Display Message with Toast
        val toast = Toast.makeText(applicationContext, "Hello Students",
Toast.LENGTH_LONG)
        toast.show()

        //Set Gravity of Display Message with Toast
        val myToast = Toast.makeText(applicationContext,"toast message
with gravity",Toast.LENGTH_SHORT)
        myToast.setGravity(Gravity.LEFT,200,200)
        myToast.show()
    }
}
}

```

3) TextView and EditText

activity_main.xml file

```

<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="12dp"
        android:text="TextView and EditText"
        android:gravity="center"
        android:textAppearance="@style/Base.TextAppearance.AppCompat.Large"/>

    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/textView1"
        android:layout_marginTop="90dp"
        android:layout_marginLeft="20dp"
        android:layout_marginStart="20dp"
        android:textSize="1sp"
        android:text="Provide Input" />

    <TextView
        android:id="@+id/textView3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"

```

```
        android:layout_below="@+id/textView2"
        android:layout_marginTop="50dp"
        android:layout_marginLeft="20dp"
        android:layout_marginStart="20dp"
        android:textSize="14sp"
        android:text="Display Output" />
```

<EditText

```
        android:id="@+id/editText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBaseline="@+id/textView2"
        android:layout_alignBottom="@+id/textView2"
        android:layout_alignParentEnd="true"
        android:layout_alignParentRight="true"
        android:layout_marginEnd="21dp"
        android:layout_marginRight="21dp"
        android:ems="8"
        android:inputType="textPersonName"
        android:text="" />
```

<TextView

```
        android:id="@+id/textView4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/editText"
        android:layout_alignStart="@+id/editText"
        android:layout_alignTop="@+id/textView3"
        android:textSize="14sp"
        android:text="TextView" />
```

<Button

```
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/textView4"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="112dp"
        android:text="Click Button" />
```

<TextView

```
        android:id="@+id/textView5"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignEnd="@+id/editText"
        android:layout_alignRight="@+id/editText"
        android:layout_centerVertical="true"
        android:text="reset output text" />
```

```
</RelativeLayout>
```

MainActivity.kt file

```
package com.student.textviewandeditview

import android.support.v7.app.AppCompatActivity
import android.os.Bundle
import android.textEditable
import android.text.TextWatcher
import android.widget.Button
import android.widget.EditText
import android.widget.TextView
import android.widget.Toast

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        var button = findViewById<Button>(R.id.button)
        var editText = findViewById<EditText>(R.id.editText)
        var textView4 = findViewById<TextView>(R.id.textView4)
        var textView5 = findViewById<TextView>(R.id.textView5)

        button.setOnClickListener() {
            val inputValue: String = editText.text.toString()
            if (inputValue == null || inputValue.trim() == "") {
                Toast.makeText(this, "please input data, edit text cannot be blank", Toast.LENGTH_LONG).show()
            } else {
                textView4.setText(inputValue).toString()
            }
        }

        textView5.setOnClickListener() {
            if (textView4.text.toString() == null || textView4.text.toString().trim() == "") {
                Toast.makeText(this, "clicked on reset textView, \n output textView already reset", Toast.LENGTH_LONG).show()
            } else {
                textView4.setText("").toString()
            }
        }

        editText.addTextChangedListener(object: TextWatcher {
            override fun beforeTextChanged(p0: CharSequence?, p1: Int, p2: Int, p3: Int) {
                // TODO("not implemented") //To change body of created functions use File | Settings | File Templates.
                Toast.makeText(applicationContext, "executed before making any change over EditText", Toast.LENGTH_SHORT).show()
            }

            override fun onTextChanged(p0: CharSequence?, p1: Int, p2: Int, p3: Int) {
                // TODO("not implemented") //To change body of created functions use File | Settings | File Templates.
                Toast.makeText(applicationContext, "executed while making any change over EditText", Toast.LENGTH_SHORT).show()
            }
        })
    }
}
```

```
    }  
    override fun afterTextChanged(p0: Editable?) {  
        // TODO("not implemented") //To change body of created  
        functions use File | Settings | File Templates.  
        Toast.makeText(applicationContext,"executed after change made  
over EditText",Toast.LENGTH_SHORT).show()  
    }  
    })  
}  
}
```