|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| <hitle> | Escaped string, Raw string | <chare> | 1 | <pext> | <starting>Escaped string, Raw string</heading> fun main(args: Array<String>) {  val escapedString : String = "I am escaped String!\n"  var rawString :String = """This is going to be a  multi-line string and will  not have any escape sequence""";  print(escapedString)  println(rawString) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* I am escaped String! This is going to be a  multi-line string and will  not have any escape sequence </ending> | </end> |
| <hitle> | String Templates | <chare> | 1 | <pext> | <starting>String Templates</heading> fun main(args: Array<String>) {  val name : String = "Zara Ali"  println("Name - $name") // Using template with variable name  println("Name length - ${name.length}") // Using template with expression. } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Name - Zara Ali Name length - 8 </ending> | </end> |
| <hitle> | String Indexes | <chare> | 1 | <pext> | <starting>String Indexes</heading> fun main(args: Array<String>) {  val name : String = "Zara Ali"  println(name[3])  println(name[5]) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* a A </ending> | </end> |
| <hitle> | String Length | <chare> | 1 | <pext> | <starting>String Length</heading> fun main(args: Array<String>) {  val name : String = "Zara Ali"  println("The length of name :" + name.length)  println("The length of name :" + name.count())   } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* The length of name :8 The length of name :8 </ending> | </end> |
| <hitle> | String Last Index | <chare> | 1 | <pext> | <starting>String Last Index</heading> fun main(args: Array<String>) {  val name : String = "Zara Ali"  println("The index of last character in name :" + name.lastIndex) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* The index of last character in name :7 </ending> | </end> |
| <hitle> | String toUpperCase(), toLowerCase() | <chare> | 1 | <pext> | <starting>String toUpperCase(), toLowerCase()</heading> fun main(args: Array<String>) {  val name : String = "Zara Ali"  println("Upper case of name :" + name.toUpperCase())  println("Lower case of name :" + name.toLowerCase()) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Upper case of name :ZARA ALI Lower case of name :zara ali </ending> | </end> |
| <hitle> | String Concat | <chare> | 1 | <pext> | <starting>String Concat</heading> fun main(args: Array<String>) {  var firstName : String = "Zara "  var lastName : String = "Ali"  println("Full Name :" + firstName + lastName)  println("Full Name :" + firstName.plus(lastName) ) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Full Name :Zara Ali Full Name :Zara Ali </ending> | </end> |
| <hitle> | Characters from a String | <chare> | 1 | <pext> | <starting>Characters from a String</heading> fun main(args: Array<String>) {  var name : String = "Zara Ali"  println("Remove first two characters from name : " + name.drop(2))  println("Remove last two characters from name : " + name.dropLast(2)) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Remove first two characters from name : ra Ali Remove last two characters from name : Zara A </ending> | </end> |
| <hitle> | Finding a String | <chare> | 1 | <pext> | <starting>Finding a String</heading> fun main(args: Array<String>) {  var str : String = "Meditation and Yoga are synonymous with India"  println("Index of Yoga in the string - " + str.indexOf("Yoga")) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Index of Yoga in the string - 15 </ending> | </end> |
| <hitle> | Compare String | <chare> | 1 | <pext> | <starting>Compare String</heading> fun main(args: Array<String>) {  var str1 : String = "Apple"  var str2 : String = "Apple"  println(str1.compareTo(str2)) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 0 </ending> | </end> |
| <hitle> | getOrNull() String | <chare> | 1 | <pext> | <starting>getOrNull() String</heading> fun main(args: Array<String>) {  var name : String = "Zara"  println(name.getOrNull(0))  println(name.getOrNull(2))  println(name.getOrNull(100)) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Z r null </ending> | </end> |
| <hitle> | Create Array | <chare> | 1 | <pext> | <starting>Create Array</heading> val fruits = arrayOf("Apple", "Mango", "Banana", "Orange") val fruits = arrayOf<String>("Apple", "Mango", "Banana", "Orange") val num = intArrayOf(1, 2, 3, 4) Other factory methods available for creating arrays: byteArrayOf() charArrayOf() shortArrayOf() longArrayOf() </ending> | </end> |
| <hitle> | Get, Set element Array | <chare> | 1 | <pext> | <starting>Get, Set element Array</heading> fun main(args: Array<String>) {  val fruits = arrayOf<String>("Apple", "Mango", "Banana", "Orange")  println( fruits [0])  println( fruits [3]) } fun main(args: Array<String>) {  val fruits = arrayOf<String>("Apple", "Mango", "Banana", "Orange")  println( fruits.get(0))  println( fruits.get(3))  // Set the value at 3rd index  fruits.set(3, "Guava")  println( fruits.get(3))  } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Apple Orange </ending> | </end> |
| <hitle> | Array Length | <chare> | 1 | <pext> | <starting>Array Length</heading> fun main(args: Array<String>) {  val fruits = arrayOf<String>("Apple", "Mango", "Banana", "Orange")  println( "Size of fruits array " + fruits.size ) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Size of fruits array 4 </ending> | </end> |
| <hitle> | Loop Array | <chare> | 1 | <pext> | <starting>Loop Array</heading> fun main(args: Array<String>) {  val fruits = arrayOf<String>("Apple", "Mango", "Banana", "Orange")  for( item in fruits ){  println( item )  } } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Apple Mango Banana Orange </ending> | </end> |
| <hitle> | Check Element Exists Array | <chare> | 1 | <pext> | <starting>Check Element Exists Array</heading> fun main(args: Array<String>) {  val fruits = arrayOf<String>("Apple", "Mango", "Banana", "Orange")  if ("Mango" in fruits){  println( "Mango exists in fruits" )  }else{  println( "Mango does not exist in fruits" )  } } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Mango exists in fruits </ending> | </end> |
| <hitle> | Distinct Array | <chare> | 1 | <pext> | <starting>Distinct Array</heading> fun main(args: Array<String>) {  val fruits = arrayOf<String>("Apple", "Mango", "Banana", "Orange", "Apple")  val distinct = fruits.distinct()  for( item in distinct ){  println( item )  } } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Apple Mango Banana Orange </ending> | </end> |
| <hitle> | Drop Elements Array | <chare> | 1 | <pext> | <starting>Drop Elements Array</heading> fun main(args: Array<String>) {  val fruits = arrayOf<String>("Apple", "Mango", "Banana", "Orange", "Apple")  val result = fruits.drop(2) // drops first two elements.  for( item in result ){  println( item )  } } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Banana Orange Apple </ending> | </end> |
| <hitle> | Check Empty Array | <chare> | 1 | <pext> | <starting>Check Empty Array</heading> fun main(args: Array<String>) {  val fruits = arrayOf<String>()  println( "Array is empty : " + fruits.isEmpty()) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Array is empty : true </ending> | </end> |
| <hitle> | Create Range rangeTo() | <chare> | 1 | <pext> | <starting>Create Range</heading> fun main(args: Array<String>) {  for ( num in 1.rangeTo(4) ) {  println(num)  } } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1 2 3 4 </ending> | </end> |
| <hitle> | Create Range .. | <chare> | 1 | <pext> | <starting>Create Range ..</heading> fun main(args: Array<String>) {  for ( num in 1..4 ) {  println(num)  } } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1 2 3 4 </ending> | </end> |
| <hitle> | Create Range downTo() | <chare> | 1 | <pext> | <starting>Create Range downTo()</heading> fun main(args: Array<String>) {  for ( num in 4 downTo 1 ) {  println(num)  } } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 4 3 2 1 </ending> | </end> |
| <hitle> | Range step() | <chare> | 1 | <pext> | <starting>Range step()</heading> fun main(args: Array<String>) {  for ( num in 1..10 step 2 ) {  println(num)  } } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1 3 5 7 9 </ending> | </end> |
| <hitle> | range of Characters | <chare> | 1 | <pext> | <starting>range of Characters</heading> fun main(args: Array<String>) {  for ( ch in 'a'..'d' ) {  println(ch)  } } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* a b c d </ending> | </end> |
| <hitle> | range reversed() | <chare> | 1 | <pext> | <starting>range reversed()</heading> fun main(args: Array<String>) {  for ( num in (1..5).reversed() ) {  println(num)  } } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 5 4 3 2 1 </ending> | </end> |
| <hitle> | range until() | <chare> | 1 | <pext> | <starting>range until()</heading> fun main(args: Array<String>) {  for ( num in 1 until 5 ) {  println(num)  } } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1 2 3 4 </ending> | </end> |
| <hitle> | range last, first, step | <chare> | 1 | <pext> | <starting>range last, first, step</heading> fun main(args: Array<String>) {  println((5..10).first)  println((5..10 step 2).step)  println((5..10).reversed().last) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 5 2 5 </ending> | </end> |
| <hitle> | Filtering Ranges | <chare> | 1 | <pext> | <starting>Filtering Ranges</heading> fun main(args: Array<String>) {  val a = 1..10  val f = a.filter { T -> T % 2 == 0 }  println(f) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* [2, 4, 6, 8, 10] </ending> | </end> |
| <hitle> | Distinct Range | <chare> | 1 | <pext> | <starting>Distinct Range</heading> fun main(args: Array<String>) {  val a = listOf(1, 1, 2, 4, 4, 6, 10)  println(a.distinct()) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* [1, 2, 4, 6, 10] </ending> | </end> |
| <hitle> | range min, max, sum, average, count | <chare> | 1 | <pext> | <starting>range min, max, sum, average, count</heading> fun main(args: Array<String>) {  val a = 1..10  println(a.min())  println(a.max())  println(a.sum())  println(a.average())  println(a.count()) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1 10 55 5.5 10 </ending> | </end> |
| <hitle> | Unit-returning Functions | <chare> | 1 | <pext> | <starting>Unit-returning Functions</heading> fun sumTwo(a:Int, b:Int):Unit{  val x = a + b  println( x ) } fun sumTwo(a:Int, b:Int){  val x = a + b  println( x ) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  </ending> | </end> |
| <hitle> | Recursive Function | <chare> | 1 | <pext> | <starting>Recursive Function</heading> fun main(args: Array<String>) {  val a = 4  val result = factorial(a)  println( result ) } fun factorial(a:Int):Int{  val result:Int  if( a <= 1){  result = a  }else{  result = a\*factorial(a-1)  }  return result } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 24 </ending> | </end> |
| <hitle> | Tail Recursion Function | <chare> | 1 | <pext> | <starting>Tail Recursion Function</heading> fun main(args: Array<String>) {  val a = 4  val result = factorial(a)  println( result ) } fun factorial(a: Int, accum: Int = 1): Int {  val result = a \* accum  return if (a <= 1) {  result  } else {  factorial(a - 1, result)  } } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 24 </ending> | </end> |
| <hitle> | Higher-Order Function sum | <chare> | 1 | <pext> | <starting>Higher-Order Function sum</heading> fun main(args: Array<String>) {  val result = calculate(4, 5, ::sum)   println( result ) } fun sum(a: Int, b: Int) = a + b  fun calculate(a: Int, b: Int, operation:(Int, Int) -> Int): Int {  return operation(a, b)  } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 9 </ending> | </end> |
| <hitle> | Higher-Order Function square | <chare> | 1 | <pext> | <starting>Higher-Order Function square</heading> fun main(args: Array<String>) {   val func = operation()   println( func(4) ) } fun square(x: Int) = x \* x fun operation(): (Int) -> Int {  return ::square  } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  </ending> | </end> |
| <hitle> | Lambda 람다총정리 | <chare> | 1 | <pext> | <starting>Lambda 람다총정리</heading> fun main(args: Array<String>) {   val upperCase = { str: String -> str.toUpperCase() }   println( upperCase("hello, world!") ) //HELLO, WORLD! }  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* {x,y -> x+y}   val multi = {x:Int,y:Int -> x\*y} println(multi(10,20))  val multi2:(Int,Int)->Int = {x:Int,y:Int ->  println("x\*y")  x\*y //마지막표현식이 반환됨 } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* //생략되지 않은 전체표현 val multi:(Int,Int)->Int = {x:Int,y:Int -> x\*y} //선언자료형생략 val multi = {x:Int,y:Int -> x\*y} //람다식 매개변수자료형의 생략 val multi:(Int,Int)->Int = {x,y -> x\*y} //에로!!추론 불가 val multi:{X,y -> x\*y} \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* //반환값이 없는 람다식 val out:()->Unit = {println("Hello World!")} //추론이 가능하므로 val out={println("Hello World!")}와 같이 생략가능 out() //Hello World! 함수처럼사용가능 //람다식을 매개변수와 인자로 사용 result = highOrder({x,y->x+y}, 10, 20) println(result)//30 fun highOrder(calculate:(Int,Int)->Int, a:Int, b:Int):Int {  return calculate(a,b) } //람다식을 값에 의한 호출 val reslut = callByuValue(lambda()) fun callByValue(b:Boolean) {  println("callByValue function")  return b } val lambda:()->Boolean = {  prinln("lambda function")  true } //람다식을 이름에 의한 호출 val result2 = callByName(otherLambda) fun callByName(b:()->Boolean):Boolean {  println("callByName function")  return b() } val otherLambda:()->Boolean = {  println("otherLambda function")  true } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* //함수는 람다처름 쓸수 없다 val res1 = funParam(3,2,sum)//오유 val res1 = funParam(3,2,::sum) prinln(res1) //5 fun sum(a:Int,b:Int)=a+b fun funcParam(a:Int,b:Int, c:(Int,Int)->Int):Int{  return c(a,b) } //인자가 없는 함수 hello(::text) //Hi! Hello World fun text(a:String,b:String) = "Hi! $a $b" fun hello(body:(String,String)->String):Uint{  prinln(body("Hello","World")) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* //매개변수가 없는 경우 noParam({"Hello World!"}) noParam{"Hello World!!!"}//우와 동일한결과, 소괄호 생략가능 //매개변수가 없는 람다식함수가 noParam함수의 매개변수 out로 지정됨 fun noParam(out:()->String)=println(out()) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* //매개변수가 1개 있는 경우 oneParam({a->"Hello World! $a"}) //Hello World! OneParam oneParam{a->"Hello World! $a"}//우와 동일 oneParam{"Hello World! $it"}//우와 동일 fun oneParam(out:(String)->String){  println(out("OneParam")) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* //매개변수생략 moreParam{a,b->"Hello World! $a $b"}//매개변수명 생략불가 fun moreParam(out:(String,String)->String){  prinln(out("P1","P2")) } moreParam{\_,b->"Hello World! $b"}//첫파라메터는 사용하지 않고 생략 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* //마지막 인자가 람다식인 경우 분리가능 withArgs("P1","P2",{a,b->"Hello World! $a $b"}) withArgs("P1","P2"){a,b->"Hello World! $a $b"}//우와 동일 fun withArgs(a:String,b:String,out:(String,String)->String){  println(out(a,b)) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* //두개의 람다식을 가진 함수 twoLambda({a,b->"First $a $b"}, {"Second $it"}) twoLambda({a,b->"First $a $b"}) {"Second $it"}) //우와 동일 fun twoLambda(first:(String,String)->String, second:(String)->String){  println(first("P1","P2"))  println(second("OneP")) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* //람다식에서 return retFunc() inline fun inlineLambda(a:Int,b:Int,out:(Int,Int)->Unit){  out(a,b) } fun retFunc(){  println("start of retFunc")  inlineLambda(13,3){a,b  val result=a+b  if(result>10) return //10보다 크면 이함수를 빠져나감  println("result:$result") //10보다 크면 이문장에 도달못함  }  println("end of retFunc") } //출력결과 start of retFunc \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* //람다식에서 라벨과 함께 return retFunc() fun inlineLambda(a:Int, b:Int, out:(Int,Int)->Unit) {  out(a,b) } fun retFunc() {  println("start of retFunc")  inlineLambda(13,3) lit@{a,b ->  val result = a+b  if(result>10) return @lit //라벨lit를 사용한 불록의 끝부분으로 반환  println("result:$result")  }  println("end of retFunc") } //출력결과 start of retFunc end of retFunc //return라벨생략 fun retFunc() {  println("start of retFunc")  inlineLambda(13,3) {a,b ->  val result = a+b  if(result>10) return @inlineLambda  println("result:$result")  }  println("end of retFunc") } </ending> | </end> |
| <hitle> | Inline Function | <chare> | 1 | <pext> | <starting>Inline Function</heading> fun main(args: Array<String>) {   myFunction({println("Inline function parameter")}) } inline fun myFunction(function:()-> Unit){  println("I am inline function - A")  function()  println("I am inline function - B") }  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* I am inline function - A Inline function parameter I am inline function - B </ending> | </end> |
| <hitle> | if...else Expression | <chare> | 1 | <pext> | <starting>if...else Expression</heading>  if (age > 18) {  print("Adult")  } else {  print("Minor")  } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  val result = if (age > 18) {  "Adult"  } else {  "Minor"  } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  val result = if (age > 18) "Adult" else "Minor" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  val result = if (age > 18) {  println("Given condition is true")  "Adult"  } else {  println("Given condition is false")  "Minor"  } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  val result = if (age > 19) {  "Adult"  } else if ( age > 12 && age < 20 ){  "Teen"  } else {  "Minor"  } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  val result = if (age > 12) {  if ( age > 12 && age < 20 ){  "Teen"  }else{  "Adult"  }  } else {  "Minor"  } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* </ending> | </end> |
| <hitle> | when | <chare> | 1 | <pext> | <starting>when</heading>  when (day) {  1 -> println("Monday")  2 -> println("Tuesday")  3 -> println("Wednesday")  4 -> println("Thursday")  5 -> println("Friday")  6 -> println("Saturday")  7 -> println("Sunday")  else -> println("Invalid day.")  } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  when (day) {  1, 2, 3, 4, 5 -> println("Weekday")  else -> println("Weekend")  } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  when (day) {  in 1..5 -> println("Weekday")  else -> println("Weekend")  } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  when (x) {  (y+z) -> print("y + z = x = $x")  else -> print("Condition is not satisfied")  } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  when (day) {  1 -> {  println("First day of the week")  println("Monday")  }  2 -> {  println("Second day of the week")  println("Tuesday")  }  3 -> {  println("Third day of the week")  println("Wednesday")  }  4 -> println("Thursday")  5 -> println("Friday")  6 -> println("Saturday")  7 -> println("Sunday")  else -> println("Invalid day.")  } </ending> | </end> |
| <hitle> | loop Range | <chare> | 1 | <pext> | <starting>loop Range</heading> fun main(args: Array<String>) {  for (item in 1..5) {  println(item)  } } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* fun main(args: Array<String>) {  for (item in 5 downTo 1 step 2) {  println(item)  } } </ending> | </end> |
| <hitle> | loop Array Indices | <chare> | 1 | <pext> | <starting>loop Array Indices</heading> fun main(args: Array<String>) {  var fruits = arrayOf("Orange", "Apple", "Mango", "Banana")  for (index in fruits.indices) {  println(fruits[index])  } } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Orange Apple Mango Banana </ending> | </end> |
| <hitle> | While Loop | <chare> | 1 | <pext> | <starting>While Loop</heading> fun main(args: Array<String>) {  var i = 5;  while (i > 0) {  println(i)  i--  } } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  </ending> | </end> |
| <hitle> | do...while Loop | <chare> | 1 | <pext> | <starting>do...while Loop</heading> fun main(args: Array<String>) {  var i = 5;  do{  println(i)  i--  }while(i > 0) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  </ending> | </end> |
| <hitle> | break | <chare> | 1 | <pext> | <starting>break</heading> fun main(args: Array<String>) {  var i = 0;  while (i++ < 100) {  println(i)  if( i == 3 ){  break  }  } } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1 2 3 </ending> | </end> |
| <hitle> | Labeled Break | <chare> | 1 | <pext> | <starting>Labeled Break</heading> fun main(args: Array<String>) {  outerLoop@ for (i in 1..3) {   innerLoop@ for (j in 1..3) {   println("i = $i and j = $j")   if (i == 2){   break@outerLoop  }   }   }  } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* i = 1 and j = 1 i = 1 and j = 2 i = 1 and j = 3 i = 2 and j = 1 </ending> | </end> |
| <hitle> | Continue | <chare> | 1 | <pext> | <starting>Continue</heading> fun main(args: Array<String>) {  var i = 0;  while (i++ < 6) {  if( i == 3 ){  continue  }  println(i)  } } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1 2 4 5 6 </ending> | </end> |
| <hitle> | Labeled Continue | <chare> | 1 | <pext> | <starting>Labeled Continue</heading> fun main(args: Array<String>) {  outerLoop@ for (i in 1..3) {   innerLoop@ for (j in 1..3) {   if (i == 2){   continue@outerLoop  }  println("i = $i and j = $j")   }   }  } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* i = 1 and j = 1 i = 1 and j = 2 i = 1 and j = 3 i = 3 and j = 1 i = 3 and j = 2 i = 3 and j = 3 </ending> | </end> |
| <hitle> | Immutable Collection | <chare> | 1 | <pext> | <starting>Immutable Collection</heading> val numbers = listOf("one", "two", "three", "four") \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  </ending> | </end> |
| <hitle> | Mutable Collection | <chare> | 1 | <pext> | <starting>Mutable Collection</heading>  val numbers = mutableListOf("one", "two", "three", "four")  numbers.add("five") \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  </ending> | </end> |
| <hitle> | create List | <chare> | 1 | <pext> | <starting>create List</heading> fun main() {  val theList = listOf("one", "two", "three", "four")  println(theList)  val theMutableList = mutableListOf("one", "two", "three", "four")  println(theMutableList) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* [one, two, three, four] [one, two, three, four] </ending> | </end> |
| <hitle> | loop List iterator | <chare> | 1 | <pext> | <starting>loop List iterator</heading> fun main() {  val theList = listOf("one", "two", "three", "four")  val itr = theList.listIterator()   while (itr.hasNext()) {  println(itr.next())  } } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* one two three four </ending> | </end> |
| <hitle> | loop List for | <chare> | 1 | <pext> | <starting>loop List for</heading> fun main() {  val theList = listOf("one", "two", "three", "four")  for (i in theList.indices) {  println(theList[i])  } } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  </ending> | </end> |
| <hitle> | loop List forEach | <chare> | 1 | <pext> | <starting>loop List forEach</heading> fun main() {  val theList = listOf("one", "two", "three", "four")  theList.forEach { println(it) } } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  </ending> | </end> |
| <hitle> | size List | <chare> | 1 | <pext> | <starting>size List</heading> fun main() {  val theList = listOf("one", "two", null, "four", "five")  println("Size of the list " + theList.size) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Size of the list 5 </ending> | </end> |
| <hitle> | in Operator List | <chare> | 1 | <pext> | <starting>in Operator List</heading> fun main() {  val theList = listOf("one", "two", "three", "four")  if("two" in theList){  println(true)  }else{  println(false)  } } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* true </ending> | </end> |
| <hitle> | contain() List | <chare> | 1 | <pext> | <starting>contain() List</heading> fun main() {  val theList = listOf("one", "two", "three", "four")  if(theList.contains("two")){  println(true)  }else{  println(false)  } } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* true </ending> | </end> |
| <hitle> | isEmpty() List | <chare> | 1 | <pext> | <starting>isEmpty() List</heading> fun main() {  val theList = listOf("one", "two", "three", "four")  if(theList.isEmpty()){  println(true)  }else{  println(false)  } } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* false </ending> | </end> |
| <hitle> | indexOf() List | <chare> | 1 | <pext> | <starting>indexOf() List</heading> fun main() {  val theList = listOf("one", "two", "three", "four")  println("Index of 'two' : " + theList.indexOf("two")) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Index of 'two' : 1 </ending> | </end> |
| <hitle> | get() List | <chare> | 1 | <pext> | <starting>get() List</heading> fun main() {  val theList = listOf("one", "two", "three", "four")  println("Element at 3rd position " + theList.get(2)) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Element at 3rd position three </ending> | </end> |
| <hitle> | List Addition | <chare> | 1 | <pext> | <starting>List Addition</heading> fun main() {  val firstList = listOf("one", "two", "three")  val secondList = listOf("four", "five", "six")  val resultList = firstList + secondList  println(resultList) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* [one, two, three, four, five, six] </ending> | </end> |
| <hitle> | List Subtraction | <chare> | 1 | <pext> | <starting>List Subtraction</heading> fun main() {  val firstList = listOf("one", "two", "three")  val secondList = listOf("one", "five", "six")  val resultList = firstList - secondList  println(resultList) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* [two, three] </ending> | </end> |
| <hitle> | Slicing a List | <chare> | 1 | <pext> | <starting>Slicing a List</heading> fun main() {  val theList = listOf("one", "two", "three", "four", "five")  val resultList = theList.slice( 2..4)  println(resultList) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* [three, four, five] </ending> | </end> |
| <hitle> | Removing null a List | <chare> | 1 | <pext> | <starting>Removing null a List</heading> fun main() {  val theList = listOf("one", "two", null, "four", "five")  val resultList = theList.filterNotNull()  println(resultList) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* [one, two, four, five] </ending> | </end> |
| <hitle> | Filter List | <chare> | 1 | <pext> | <starting>Filter List</heading> fun main() {  val theList = listOf(10, 20, 30, 31, 40, 50, -1, 0)  val resultList = theList.filter{ it > 30}  println(resultList) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* [31, 40, 50] </ending> | </end> |
| <hitle> | Drop elements List | <chare> | 1 | <pext> | <starting>Drop elements List</heading> fun main() {  val theList = listOf(10, 20, 30, 31, 40, 50, -1, 0)  val resultList = theList.drop(3)  println(resultList) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* [31, 40, 50, -1, 0] </ending> | </end> |
| <hitle> | group List | <chare> | 1 | <pext> | <starting>group List</heading> fun main() {  val theList = listOf(10, 12, 30, 31, 40, 9, -3, 0)  val resultList = theList.groupBy{ it % 3}  println(resultList) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* {1=[10, 31, 40], 0=[12, 30, 9, -3, 0]} </ending> | </end> |
| <hitle> | Mapping List | <chare> | 1 | <pext> | <starting>Mapping List</heading> fun main() {  val theList = listOf(10, 12, 30, 31, 40, 9, -3, 0)  val resultList = theList.map{ it / 3 }  println(resultList) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* [3, 4, 10, 10, 13, 3, -1, 0] </ending> | </end> |
| <hitle> | Chunking List chunked() | <chare> | 1 | <pext> | <starting>Chunking List chunked()</heading> fun main() {  val theList = listOf(10, 12, 30, 31, 40, 9, -3, 0)  val resultList = theList.chunked(3)  println(resultList) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* [[10, 12, 30], [31, 40, 9], [-3, 0]] </ending> | </end> |
| <hitle> | Windowing List windowed() | <chare> | 1 | <pext> | <starting>Windowing List windowed()</heading> fun main() {  val theList = listOf(10, 12, 30, 31, 40, 9, -3, 0)  val resultList = theList.windowed(3)  println(resultList) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* [[10, 12, 30], [12, 30, 31], [30, 31, 40], [31, 40, 9], [40, 9, -3], [9, -3, 0]] </ending> | </end> |
| <hitle> | Windowing List by Step windowed() | <chare> | 1 | <pext> | <starting>Windowing List by Step windowed()</heading> fun main() {  val theList = listOf(10, 12, 30, 31, 40, 9, -3, 0)  val resultList = theList.windowed(3, 3)  println(resultList) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* [[10, 12, 30], [31, 40, 9]] </ending> | </end> |
| <hitle> | mutable List | <chare> | 1 | <pext> | <starting>mutable List</heading> fun main() {  val theList = mutableSetOf(10, 20, 30)  theList.add(40)  theList.add(50)  println(theList)  theList.remove(10)  theList.remove(30)  println(theList) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* [10, 20, 30, 40, 50] [20, 40, 50] </ending> | </end> |
| <hitle> | create Set | <chare> | 1 | <pext> | <starting>create Set</heading> fun main() {  val theSet = setOf("one", "two", "three", "four")  println(theSet)  val theMutableSet = mutableSetOf("one", "two", "three", "four")  println(theMutableSet) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* [one, two, three, four] [one, two, three, four] </ending> | </end> |
| <hitle> | loop Set Iterator | <chare> | 1 | <pext> | <starting>loop Set Iterator</heading> fun main() {  val theSet = setOf("one", "two", "three", "four")  val itr = theSet.asIterable().iterator()  while (itr.hasNext()) {  println(itr.next())  } } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* one two three four </ending> | </end> |
| <hitle> | loop Set for | <chare> | 1 | <pext> | <starting>loop Set for</heading> fun main() {  val theSet = setOf("one", "two", "three", "four")  for (i in theSet.indices) {  println(theSet.elementAt(i))  } } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* one two three four </ending> | </end> |
| <hitle> | loop Set forEach | <chare> | 1 | <pext> | <starting>loop Set forEach</heading> fun main() {  val theSet = setOf("one", "two", "three", "four")  theSet.forEach { println(it) } } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* one two three four </ending> | </end> |
| <hitle> | size Set | <chare> | 1 | <pext> | <starting>size Set</heading> fun main() {  val theSet = setOf("one", "two", null, "four", "five")  println("Size of the Set " + theSet.size) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Size of the Set 5 </ending> | </end> |
| <hitle> | in Operator Set | <chare> | 1 | <pext> | <starting>in Operator Set</heading> fun main() {  val theSet = setOf("one", "two", "three", "four")  if("two" in theSet){  println(true)  }else{  println(false)  } } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* true </ending> | </end> |
| <hitle> | contain() Set | <chare> | 1 | <pext> | <starting>contain() Set</heading> fun main() {  val theSet = setOf("one", "two", "three", "four")  if(theSet.contains("two")){  println(true)  }else{  println(false)  } } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* true </ending> | </end> |
| <hitle> | isEmpty() Set | <chare> | 1 | <pext> | <starting> isEmpty() Set</heading> fun main() {  val theSet = setOf("one", "two", "three", "four")  if(theSet.isEmpty()){  println(true)  }else{  println(false)  } } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* false </ending> | </end> |
| <hitle> | elementAt() Set | <chare> | 1 | <pext> | <starting>elementAt() Set</heading> fun main() {  val theSet = setOf("one", "two", "three", "four")  println("Element at 3rd position " + theSet.elementAt(2)) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Element at 3rd position three </ending> | </end> |
| <hitle> | indexOf() Set | <chare> | 1 | <pext> | <starting>indexOf() Set</heading> fun main() {  val theSet = setOf("one", "two", "three", "four")  println("Index of 'two' - " + theSet.indexOf("two")) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Index of 'two' - 1 </ending> | </end> |
| <hitle> | Set Addition | <chare> | 1 | <pext> | <starting>Set Addition</heading> fun main() {  val firstSet = setOf("one", "two", "three")  val secondSet = setOf("one", "four", "five", "six")  val resultSet = firstSet + secondSet  println(resultSet) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* [one, two, three, four, five, six] </ending> | </end> |
| <hitle> | Set Subtraction | <chare> | 1 | <pext> | <starting>Set Subtraction</heading> fun main() {  val firstSet = setOf("one", "two", "three")  val secondSet = setOf("one", "five", "six")  val resultSet = firstSet - secondSet  println(resultSet) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* [two, three] </ending> | </end> |
| <hitle> | Removing null a Set | <chare> | 1 | <pext> | <starting>Removing null a Set</heading> fun main() {  val theSet = setOf("one", "two", null, "four", "five")  val resultSet = theSet.filterNotNull()  println(resultSet) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* [one, two, four, five] </ending> | </end> |
| <hitle> | Sorting Set | <chare> | 1 | <pext> | <starting>Sorting Set</heading> fun main() {  val theSet = setOf(10, 20, 30, 31, 40, 50, -1, 0)  var resultSet = theSet.sorted()  println(resultSet)  resultSet = theSet.sortedDescending()  println(resultSet) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* [-1, 0, 10, 20, 30, 31, 40, 50] [50, 40, 31, 30, 20, 10, 0, -1] </ending> | </end> |
| <hitle> | Filtering Set | <chare> | 1 | <pext> | <starting>Filtering Set</heading> fun main() {  val theSet = setOf(10, 20, 30, 31, 40, 50, -1, 0)  val resultSet = theSet.filter{ it > 30}  println(resultSet) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* [31, 40, 50] </ending> | </end> |
| <hitle> | Drop Elements Set | <chare> | 1 | <pext> | <starting>Drop Elements Set</heading> fun main() {  val theSet = setOf(10, 20, 30, 31, 40, 50, -1, 0)  val resultSet = theSet.drop(3)  println(resultSet) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* [31, 40, 50, -1, 0] </ending> | </end> |
| <hitle> | Grouping Set | <chare> | 1 | <pext> | <starting>Grouping Set</heading> fun main() {  val theSet = setOf(10, 12, 30, 31, 40, 9, -3, 0)  val resultSet = theSet.groupBy{ it % 3}  println(resultSet) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* {1=[10, 31, 40], 0=[12, 30, 9, -3, 0]} </ending> | </end> |
| <hitle> | Mapping Set | <chare> | 1 | <pext> | <starting>Mapping Set</heading> fun main() {  val theSet = setOf(10, 12, 30, 31, 40, 9, -3, 0)  val resultSet = theSet.map{ it / 3 }  println(resultSet) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* [3, 4, 10, 10, 13, 3, -1, 0] </ending> | </end> |
| <hitle> | Chunking Set chunked() | <chare> | 1 | <pext> | <starting>Chunking Set chunked()</heading> fun main() {  val theSet = setOf(10, 12, 30, 31, 40, 9, -3, 0)  val resultSet = theSet.chunked(3)  println(resultSet) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* [[10, 12, 30], [31, 40, 9], [-3, 0]] </ending> | </end> |
| <hitle> | Windowing Set windowed() | <chare> | 1 | <pext> | <starting>Windowing Set windowed()</heading> fun main() {  val theSet = setOf(10, 12, 30, 31, 40, 9, -3, 0)  val resultSet = theSet.windowed(3)  println(resultSet) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* [[10, 12, 30], [12, 30, 31], [30, 31, 40], [31, 40, 9], [40, 9, -3], [9, -3, 0]] </ending> | </end> |
| <hitle> | Windowing Set by step windowed() | <chare> | 1 | <pext> | <starting>Windowing Set by step windowed()</heading> fun main() {  val theSet = setOf(10, 12, 30, 31, 40, 9, -3, 0)  val resultSet = theSet.windowed(3, 3)  println(resultSet) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* [[10, 12, 30], [31, 40, 9]] </ending> | </end> |
| <hitle> | mutable Set | <chare> | 1 | <pext> | <starting>mutable Set</heading> fun main() {  val theSet = mutableSetOf(10, 20, 30)  theSet.add(40)  theSet.add(50)  println(theSet)  theSet.remove(10)  theSet.remove(30)  println(theSet) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* [10, 20, 30, 40, 50] [20, 40, 50] </ending> | </end> |
| <hitle> | create Map | <chare> | 1 | <pext> | <starting>create Map</heading> fun main() {  val theMap = mapOf("one" to 1, "two" to 2, "three" to 3, "four" to 4)  println(theMap)  val theMutableMap = mutableSetOf("one" to 1, "two" to 2, "three" to 3, "four" to 4)  println(theMutableMap) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* {one=1, two=2, three=3, four=4} [(one, 1), (two, 2), (three, 3), (four, 4)] </ending> | </end> |
| <hitle> | Create Map by HashMap | <chare> | 1 | <pext> | <starting>Create Map by HashMap</heading> fun main() {  val theMap = HashMap<String, Int>()  theMap["one"] = 1  theMap["two"] = 2  theMap["three"] = 3  theMap["four"] = 4  println(theMap) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* {four=4, one=1, two=2, three=3} </ending> | </end> |
| <hitle> | Create Map by Pair() | <chare> | 1 | <pext> | <starting>Create Map by Pair()</heading> fun main() {  val theMap = mapOf(Pair("one", 1), Pair("two", 2), Pair("three", 3))  println(theMap) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* {one=1, two=2, three=3} </ending> | </end> |
| <hitle> | Map Properties | <chare> | 1 | <pext> | <starting>Map Properties</heading> fun main() {  val theMap = mapOf("one" to 1, "two" to 2, "three" to 3, "four" to 4)  println("Entries: " + theMap.entries)  println("Keys:" + theMap.keys)  println("Values:" + theMap.values) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Entries: [one=1, two=2, three=3, four=4] Keys:[one, two, three, four] Values:[1, 2, 3, 4] </ending> | </end> |
| <hitle> | loop Map Iterator | <chare> | 1 | <pext> | <starting>loop Map Iterator</heading> fun main() {  val theMap = mapOf("one" to 1, "two" to 2, "three" to 3, "four" to 4)  val itr = theMap.keys.iterator()  while (itr.hasNext()) {  val key = itr.next()  val value = theMap[key]  println("${key}=$value")  } } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* one=1 two=2 three=3 four=4 </ending> | </end> |
| <hitle> | loop Map for | <chare> | 1 | <pext> | <starting>loop Map for</heading> fun main() {  val theMap = mapOf("one" to 1, "two" to 2, "three" to 3, "four" to 4)  for ((k, v) in theMap) {  println("$k = $v")  } } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* one = 1 two = 2 three = 3 four = 4 </ending> | </end> |
| <hitle> | loop Map forEach | <chare> | 1 | <pext> | <starting>loop Map forEach</heading> fun main() {  val theMap = mapOf("one" to 1, "two" to 2, "three" to 3, "four" to 4)  theMap.forEach {   k, v -> println("Key = $k, Value = $v")   } } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Key = one, Value = 1 Key = two, Value = 2 Key = three, Value = 3 Key = four, Value = 4 </ending> | </end> |
| <hitle> | size Map | <chare> | 1 | <pext> | <starting>size Map</heading> fun main() {  val theMap = mapOf("one" to 1, "two" to 2, "three" to 3, "four" to 4)  println("Size of the Map " + theMap.size)  println("Size of the Map " + theMap.count()) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Size of the Map 4 Size of the Map 4 </ending> | </end> |
| <hitle> | Map containsKey(), containsValue() | <chare> | 1 | <pext> | <starting>Map containsKey(), containsValue()</heading> fun main() {  val theMap = mapOf("one" to 1, "two" to 2, "three" to 3, "four" to 4)  if(theMap.containsKey("two")){  println(true)  }else{  println(false)  }  if(theMap.containsValue("two")){  println(true)  }else{  println(false)  }  } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* true false </ending> | </end> |
| <hitle> | isEmpty() Map | <chare> | 1 | <pext> | <starting>isEmpty() Map</heading> fun main() {  val theMap = mapOf("one" to 1, "two" to 2, "three" to 3, "four" to 4)  if(theMap.isEmpty()){  println(true)  }else{  println(false)  } } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* false </ending> | </end> |
| <hitle> | get() Map | <chare> | 1 | <pext> | <starting>get() Map</heading> fun main() {  val theMap = mapOf("one" to 1, "two" to 2, "three" to 3, "four" to 4)  println("The value for key two " + theMap.get("two"))  println("The value for key two " + theMap["two"]) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* The value for key two 2 The value for key two 2 </ending> | </end> |
| <hitle> | Map Addition | <chare> | 1 | <pext> | <starting>Map Addition</heading> fun main() {  val firstMap = mapOf("one" to 1, "two" to 2, "three" to 3)  val secondMap = mapOf("one" to 10, "four" to 4)  val resultMap = firstMap + secondMap  println(resultMap) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* {one=10, two=2, three=3, four=4} </ending> | </end> |
| <hitle> | Map Subtraction | <chare> | 1 | <pext> | <starting>Map Subtraction</heading> fun main() {  val theMap = mapOf("one" to 1, "two" to 2, "three" to 3)  val theKeyList = listOf("one", "four")  val resultMap = theMap - theKeyList  println(resultMap) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* {two=2, three=3} </ending> | </end> |
| <hitle> | Removing Entries from Map | <chare> | 1 | <pext> | <starting>Removing Entries from Map</heading> fun main() {  val theMap = mutableMapOf("one" to 1, "two" to 2, "three" to 3, "four" to 4)  theMap.remove( "two")  println(theMap)  theMap -= listOf("three")  println(theMap) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* {one=1, three=3, four=4} {one=1, four=4} </ending> | </end> |
| <hitle> | Sorting Map | <chare> | 1 | <pext> | <starting>Sorting Map</heading> fun main() {  val theMap = mapOf("one" to 1, "two" to 2, "three" to 3, "four" to 4)  var resultMap = theMap.toSortedMap()  println(resultMap) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* {four=4, one=1, three=3, two=2} </ending> | </end> |
| <hitle> | Filtering Map | <chare> | 1 | <pext> | <starting>Filtering Map</heading> fun main() {  val theMap = mapOf("one" to 1, "two" to 2, "three" to 3, "four" to 4)  var resultMap = theMap.filterValues{ it > 2}  println(resultMap)  resultMap = theMap.filterKeys{ it == "two"}  println(resultMap)  resultMap = theMap.filter{ it.key == "two" || it.value == 4}  println(resultMap) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* {three=3, four=4} {two=2} {two=2, four=4} </ending> | </end> |
| <hitle> | Mapping Map | <chare> | 1 | <pext> | <starting>Mapping Map</heading> fun main() {  val theMap = mapOf("one" to 1, "two" to 2, "three" to 3)  val resultMap = theMap.map{ (k, v) -> "Key is $k, Value is $v" }  println(resultMap) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* [Key is one, Value is 1, Key is two, Value is 2, Key is three, Value is 3] </ending> | </end> |
| <hitle> | Mutable Map | <chare> | 1 | <pext> | <starting>Mutable Map</heading> fun main() {  val theMap = mutableMapOf("one" to 1, "two" to 2, "three" to 3, "four" to 4)  theMap.put("four", 4)  println(theMap)  theMap["five"] = 5  println(theMap)  theMap.remove("two")  println(theMap) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* {one=1, two=2, three=3, four=4} {one=1, two=2, three=3, four=4, five=5} {one=1, three=3, four=4, five=5} </ending> | </end> |
| <hitle> | Type check, cast, is, !is | <chare> | 1 | <pext> | <starting>Type check, cast, is, !is</heading> if (obj is String) {  print(obj.length) } if (obj !is String) { // same as !(obj is String)  print("Not a String") } else {  print(obj.length) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  </ending> | </end> |
| <hitle> | groupBy() samples | <chare> | 1 | <pext> | <starting>groupBy() samples</heading> val list = listOf("후렌치파이", "치토스", "콘칩", "빼빼로") val map = list.groupBy{it -> it.length} print(map) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* {5=[후렌치파이], 3=[치토스, 빼빼로], 2=[콘칩]} \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* val list = listOf("후렌치파이", "치토스", "콘칩", "빼빼로") val map = list.groupBy({it}, {it.length}) print(map) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* {후렌치파이=[5], 치토스=[3], 콘칩=[2], 빼빼로=[3]} \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* val list = listOf("A", "AB", "C") list.groupBy(String::first).toString() //first가 같은 값으로 묶음  //출력  {A=[A, AB], C=[C]} </ending> | </end> |
| <hitle> | map() samples | <chare> | 1 | <pext> | <starting>map() samples</heading> var listInts = listOf(1,2,3,4) listInts.map { it \* it } //출력 [1,4,9,16]  val people = listOf(Person("A", 27), Person("B", 35), Person("C", 35)) listInts.map { it.name } //출력 [A, B, C] //심플한 방법 people.map(Person::name)  //30살이상의 이름 출력 people.filter { it.age >= 30 }.map(Person::name)  //가장 나이가 많은 사람들 모두 출력하기 //1. 최대값을 구하는 반복작업이 쓸데없이 많은 코드 people.filter { it.age == (people.maxBy(Person::age)!!.age) } //매번 maxBy 호출  //2. 계산이 중복되는것을 피하자. 람다안에 람다를 넣는것은 내부 로직을 매우 복잡하게 할 수 있음 val maxAge = people.maxBy(Person::age)!!.age //외부로 빼서 불필요한 반복 제거 people.filter { it.age == maxAge }  //맵 사용 //map 에는 filterKeys, mapKeys 와 filterValues, mapValues 함수가 있음 val numbers = mapOf(0 to "zero", 1 to "one") val upperNumbers = numbers.mapValues { it.value.toUpperCase() } //출력 [0=ZERO, 1=ONE] </ending> | </end> |
| <hitle> | all() samples | <chare> | 1 | <pext> | <starting>all() samples</heading> val people = listOf(Person("A", 27), Person("B", 35), Person("C", 35)) val canBeInClub27 = { p: Person -> p.age <= 27 } people.all(canBeInClub27)  //출력 false //모든 값이 27 이하가 아니기때문에 false 출력 </ending> | </end> |
| <hitle> | any() samples | <chare> | 1 | <pext> | <starting>any() samples</heading> val people = listOf(Person("A", 27), Person("B", 35), Person("C", 35)) val canBeInClub27 = { p: Person -> p.age <= 27 } people.any(canBeInClub27)  //출력 true //A가 27 으로 하나라도 만족하기 때문에 true 출력 </ending> | </end> |
| <hitle> | count() samples | <chare> | 1 | <pext> | <starting>count() samples</heading> people.count(canBeInClub27) //출력 1 //27 이하는 한개  //filter 를 사용하면 중간 컬렉션이 불필요하게 생기때문에 count 가 효율적 임 people.filter(canBeInClub27).size  </ending> | </end> |
| <hitle> | find() samples | <chare> | 1 | <pext> | <starting>find() samples</heading> // 27인 첫번째 것을 리턴하거나 없으면 null 을 리턴 people.find(canBeInClub27)  // find 에서 null 을 리턴한다는 것을 명확히 함 people.firstOrNull() </ending> | </end> |
| <hitle> | flatMap() samples | <chare> | 1 | <pext> | <starting>flatMap() samples</heading> val strings = listOf("abc", "de") strings.flatMap { it.toList() } //출력 [a, b, c, d, e] //두개의 리스트가 한개 리스트로 합침  val strings = listOf("abc", "de", "de") strings.flatMap { it.toList() }.toSet() //출력 [a, b, c, d, e] //중복제거 O  //단순히 리스트를 펼치기만 할때 strings.map{ it.toList() }.flatten() //출력 [a, b, c, d, e, d, e] //중복제거 X  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* val testList = listOf(listOf(1,2), listOf(7,8,9), mutableListOf(4,5,6)) testList.flatMap { it : List<Int> -> it } //[1,2,7,8,9,4,5,6] testList.flatMap { it : List<Int> -> it.take(1) } //[1,7,4] </ending> | </end> |
| <hitle> | Time Complexity | <chare> | 1 | <pext> | <starting>Time Complexity</heading> Default Sort: O(n\*log2(n)) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* O(n^2), On(n^3), … ----------------------- for(n1)  for(n2)  for(n3) ... \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* O(2^n), O(3^n)... --------- fun test(n) {  test(n-1) // O(n)  test(n-2) // If two recursive calls, O(2^n)  test(n-3) // If three recursive calls, O(3^n) ... \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* O(log2(n)), O(log3(n))... fun test(n) {  test(n/2) } fun test(n) {  test(n/3) } ... \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* O(n!) ------- fun test(n) {  for(n)  test(n-1) } </ending> | </end> |
| <hitle> | mapIndexed() | <chare> | 1 | <pext> | <starting>mapIndexed()/heading> ('a'..'z').mapIndexed { index, i -> Pair(index, i)} \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* [(0, a), (1, b), (2, c), (3, d), (4, e), (5, f), (6, g), (7, h), (8, i), (9, j), (10, k), (11, l), (12, m), (13, n), (14, o), (15, p), (16, q), (17, r), (18, s), (19, t), (20, u), (21, v), (22, w), (23, x), (24, y), (25, z)] </ending> | </end> |
| <hitle> | mapTo() | <chare> | 1 | <pext> | <starting>mapTo()</heading> val intList: MutableList<Int> = mutableListOf() (1..10).mapTo(intList) { it\*it } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* // [1, 4, 9, 16, 25, 36, 49, 64, 81, 100] </ending> | </end> |
| <hitle> | mapIndexedTo() | <chare> | 1 | <pext> | <starting>mapIndexedTo()</heading> var intPairList: MutableList<Pair<Int, Int>> = mutableListOf() (1..10).mapIndexedTo(intPairList) { index, it -> Pair(index, it\*it) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* // [(0, 1), (1, 4), (2, 9), (3, 16), (4, 25), (5, 36), (6, 49), (7, 64), (8, 81), (9, 100)] </ending> | </end> |
| <hitle> | map() vs flatMap() | <chare> | 1 | <pext> | <starting>map() vs flatMap()</heading> val testList = listOf("A", "B", "C") val newList1 = testList.map { "$it!" } //[A!, B!, C!] val newList2 = testList.flatMap { "$it".toList() } // [A, !, B, !, C, !] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* val vehicles = listOf(cars, bikes) val manufacturerList = vehicles.flatMap { it }.map { it.manufacturer } //vehicles의 item을 flat하게 하고 map을 이용해 manufactures만 뽑아냄 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* val ints = (1..4) // [1, 2, 3, 4] val strings = listOf("a", "b", "c", "d")  val flatMapList = ints.flatMap { int ->  strings.map { string ->  "$int$string"  } } println(flatMapList) //[1a, 1b, 1c, 1d, 2a, 2b, 2c, 2d, 3a, 3b, 3c, 3d, 4a, 4b, 4c, 4d] </ending> | </end> |
| <hitle> | groupingBy(), eachCount(), eachCountTo() | <chare> | 1 | <pext> | <starting>groupingBy(), eachCount(), eachCountTo()</heading> fun main() {  val words1 = "one two three two three".split(' ').groupBy { it }  println(words1) //{one=[one], two=[two, two], three=[three, three]}  val words2 = "one two three four".split(' ').groupBy { it }  println(words2) //{one=[one], two=[two], three=[three], four=[four]}  val duplicatedElements = words1.filter { it.value == words2[it.key] }  println(duplicatedElements) //{one=[one]} } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* fun main() {  val words = "one two three four five six seven eight nine ten".split(' ')  val frequenciesByFirstChar = words.groupingBy { it.first() }.eachCount()  println(frequenciesByFirstChar) // {o=1, t=3, f=2, s=2, e=1, n=1}  val moreWords = "eleven twelve".split(' ')  val moreFrequencies = moreWords.groupingBy { it.first() }.eachCountTo(frequenciesByFirstChar.toMutableMap())  println(moreFrequencies) // {o=1, t=4, f=2, s=2, e=2, n=1}  println((frequenciesByFirstChar).filter { it.value == moreFrequencies[it.key] }) // {o=1, f=2, s=2, n=1} } </ending> | </end> |
| <hitle> | string.format() | <chare> | 1 | <pext> | <starting>string format</heading> val PI = 3.14159265358979323 val myStr = String.format("The PI value is %.2f", PI) print(myStr)// The PI value is 3.14 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* val name = "Nathan" val myStr = String.format("My name is %s", name) print(myStr)// My name is Nathan \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* %b - Boolean %c - Character %d - Signed Integer %e - Float in Scientific Notation %f - Float in Decimal Format %g - Float in Decimal or Scientific Notation, depending on the value %h - Hashcode of the supplied argument %n - Newline separator %o - Octal Integer (base 8) %s - String %t - Date or Time %x - Hexadecimal Integer (base 16) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* val myStr = String.format("%b | %s | %d", false, "Morning", 90) print(myStr)// false | Morning | 90 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* fun main(){ // studnt details  var name = "Taylor Swift"  var roll\_no = 4676  var percentage = 96.3  var grade = 'A'  var school = "Delhi Public School"  // formating using String.format() method  var student\_data = String.format("Name: %s%nRoll No.: %d%nPercentage: %.2f%%%nGrade: %c%nSchool Name: %s", name, roll\_no, percentage, grade, school)  println(student\_data) } Name: Taylor Swift Roll No.: 4676 Percentage: 96.30% Grade: A School Name: Delhi Public School \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* fun main(){   var say\_hello = mapOf(  "Hello" to "English",  "Goedendag" to "Dutch",  "Namaste" to "Hindi",  "Bonjour" to "French",  "Hola" to "Spanish"  )  // defining default format  var string\_format = "%s\t - %s%n"  println("Greetings\tLanguage")  // printing contenets of map using default format  for((k, v) in say\_hello)  print(string\_format.format(k,v)) } Greetings Language Hello - English Goedendag - Dutch Namaste - Hindi Bonjour - French Hola - Spanish \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* fun rgb(r: Int, g: Int, b: Int) = String.format("%02X%02X%02X", r.coerceIn(0..255), g.coerceIn(0..255), b.coerceIn(0..255)) </ending> | </end> |
| <hitle> | type extention fun | <chare> | 1 | <pext> | <starting>type extention fun</heading> fun Int.abs() : Int{  return if(this < 0) -this else this  } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  </ending> | </end> |
| <hitle> | coerceIn() ensure value in minimumValue..maximumValue | <chare> | 1 | <pext> | <starting>coerceIn() ensure value in minimumValue..maximumValue</heading> rgb to Hex \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* fun rgb(r: Int, g: Int, b: Int) = String.format("%02X%02X%02X", r.coerceIn(0..255), g.coerceIn(0..255), b.coerceIn(0..255)) </ending> | </end> |
| <hitle> | split string | <chare> | 1 | <pext> | <starting>split string</heading> fun main(args: Array<String>) {  var str = "Kotlin TutorialsepTutorial KartsepExamples"  var delimiter = "sep"  val parts = str.split(delimiter)  print(parts) //[Kotlin Tutorial, Tutorial Kart, Examples] } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* fun main(args: Array<String>) {  var str = "Kotlin TutorialsepTutorialasepKartsepExamples"  var delimiter1 = "sep"  var delimiter2 = "asep"  val parts = str.split(delimiter1, delimiter2)  print(parts) //[Kotlin Tutorial, Tutorial, Kart, Examples] } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* fun main(args: Array<String>) {  var str = "Kotlin TutorialsEPTutorialaSEpKartSEpExamples"  var delimiter1 = "SEP"  var delimiter2 = "ASEP"  val parts = str.split(delimiter1, delimiter2, ignoreCase = true)  print(parts)//[Kotlin Tutorial, Tutorial, Kart, Examples] } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* fun main(args: Array<String>) {  var str = "Kotlin TutorialsepTutorialasepKartsepExamples"  val parts = str.split(Regex("sep|asep"))  print(parts) //[Kotlin Tutorial, Tutorial, Kart, Examples] } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  </ending> | </end> |
| <hitle> | substring | <chare> | 1 | <pext> | <starting>substring</heading> fun main(args: Array<String>) {  val str1 = "abcdefghij"  val startIndex = 2  val endIndex = 7  val substring = str1.subSequence(startIndex, endIndex)  println("The substring is : " + substring) //The substring is : cdefg } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* fun main(args: Array<String>) {  val str1 = "abcdefghij"  val startIndex = -2  val endIndex = 7  val substring = str1.subSequence(startIndex, endIndex)  println("The substring is : " + substring) //StringIndexOutOfBoundsException } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* fun main() {  val a = "あいうえお"  println(a.substring(1)) // いうえお  println(a.substring(2)) // うえお  println(a.substring(3)) // えお  println(a.substring(4)) // お    println(a.substring(a.length - 1)) // お  println(a.substring(a.length - 2)) // えお  println(a.substring(a.length - 3)) // うえお } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* fun main() {  val str = "Hello World !!"  println(str.substring(1..5)) //ello } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* fun main() {  val str = "Hello %World !!"  println(str.substringAfter('%'))  println(str.substringAfter("Hello"))  println(str.substringAfter('\*'))  println(str.substringAfter('\*',"Not Found")) } World !!  %World !! Hello %World !! Not Found \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* fun main() {  val str = "Hello %World !!"  println(str.substringBefore('%'))  println(str.substringBefore("Hello"))  println(str.substringBefore('\*'))  println(str.substringBefore('\*',"Not Found")) } Hello Hello %World !! Not Found \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* fun main() {  val str = "https://www.codevscolor.com/example"  println(str.substringBeforeLast('/'))  println(str.substringAfterLast('/')) } https://www.codevscolor.com example </ending> | </end> |
| <hitle> | merge, join, concatenate string joinToString() | <chare> | 1 | <pext> | <starting>merge, join, concatenate string joinToString()</heading> fun concat(s1: String, s2: String): String {  return s1 + s2 } fun concat(s1: String, s2: String): String {  return s1.plus(s2) } fun concat(s1: String, s2: String): String {  return StringBuilder(s1).append(s2).toString() } fun concat(s1: String, s2: String): String {  return "$s1 $s2" } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* fun concat(vararg string: String): String {  val sb = StringBuilder()  for (s in string) {  sb.append(s)  }  return sb.toString() } fun main() {  val s1 = "Hello"  val s2 = " "  val s3 = "World"  val result = concat(s1, s2, s3)  println(result) // Hello World } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* fun main() {  var list = listOf("Hello", "World")  val result = list.joinToString("")  println(result) // HelloWorld } </ending> | </end> |
| <hitle> | clear, remove all list clear() | <chare> | 1 | <pext> | <starting>clear, remove all list clear()</heading> fun main(args: Array<String>) {  val mutableList = mutableListOf(1, 3, 9, 16, 25)  println("List before clearing : " + mutableList)  mutableList.clear()  println("List after clearing : " + mutableList) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* List before clearing : [1, 3, 9, 16, 25] List after clearing : [] </ending> | </end> |
| <hitle> | sortedWith() compareBy() thenBy() | <chare> | 1 | <pext> | <starting>sortedWith() compareBy() thenBy()</heading> val list = listOf(7,3,5,9,1,3) list.sortedWith(Comparator<Int>{ a, b ->  when {  a > b -> 1  a < b -> -1  else -> 0  } }) [1, 3, 3, 5, 7, 9] val names = listOf("kim", "julia", "jim", "hala") names.sortedWith(Comparator<String>{ a, b ->  when {  a > b -> 1  a < b -> -1  else -> 0  } }) [hala, jim, julia, kim] names.sortedWith(Comparator<String>{ a, b ->  when {  a.length > b.length -> 1  a.length < b.length -> -1  else -> 0  } }) [kim, jim, hala, julia] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* val list = listOf("aa", "b", "bb", "a") val sorted = list.sortedWith(compareBy(  { it.length },  { it } )) println(sorted) // [a, b, aa, bb] val list = listOf("aa", "b", "bb", "a") val sorted = list.sortedWith(compareBy { it.length }) println(sorted) // [b, a, aa, bb] val list = listOf('B', 'a', 'A', 'b') val sorted = list.sortedWith(  compareBy(String.CASE\_INSENSITIVE\_ORDER) { v -> v.toString() } ) println(sorted) // [a, A, B, b] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* val sortedList = list.sortedWith(  compareBy(  { 조건 A }, // 1. 조건 A에 대한 정렬 수행 후,  { 조건 B } // 2. 1에서 정렬된 리스트를 기반으로 조건 B에 대한 정렬 수행  ) ) data class TrackingInfoCompany {  val completeYN: String, // 배송완료 시 "Y", 아니면 "N"  val time: Long // 마지막 업데이트 시간 (millisecond 단위) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* var sorted = planedit.sortedBy{it.first} var sorted = planedit.sortedWith(compareBy({ it.first }, { it.second })) var sorted = planedit.sortedWith(compareBy({ it.first }, { it.second })).reversed() \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* thenBy() 첫 조건이 동일할 때만, 다른 조건으로 비교  val dates = mutableListOf(  Date(2020, 4, 3),  Date(2021, 5, 16),  Date(2020, 4, 29)  )  dates.sortWith(compareBy<Date> { it.year }.thenBy { it.month }.thenBy { it.day })  dates.forEach { println(it) } 2020/4/03 2020/4/29 2021/5/16 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* val list = listOf("aa", "b", "bb", "a") val lengthComparator = compareBy<String> { it.length } println(list.sortedWith(lengthComparator)) // [b, a, aa, bb] val lengthThenString = lengthComparator.thenBy { it } println(list.sortedWith(lengthThenString)) // [a, b, aa, bb] val list = listOf("A", "aa", "b", "bb", "a") val lengthComparator = compareBy<String> { it.length } println(list.sortedWith(lengthComparator)) // [A, b, a, aa, bb] val lengthThenCaseInsensitive = lengthComparator  .thenBy(String.CASE\_INSENSITIVE\_ORDER) { it } println(list.sortedWith(lengthThenCaseInsensitive)) // [A, a, b, aa, bb] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* data class Man(var name:String, var age:Int) fun SortingExample() {  val rand = Random()  val randomValues = generateSequence { rand.nextInt(10 - 5) }  .take(10).toList()  println(randomValues)  println(randomValues.sorted())  val people = listOf(  Man("Adam", 36),  Man("Boris", 18),  Man("Claire", 2),  Man("Adam", 20),  Man("Jack", 20)  )  println(people)  println(people.sortedBy { it.name })  println(people.sortedWith(compareBy(Man::age, Man::name)))  println(people.sortedWith(compareBy({it.name}, {it.age})))  println(people.sortedWith(compareBy<Man>{it.age}.thenByDescending { it.name })) } </ending> | </end> |
| <hitle> | sort compare 총정리 | <chare> | 1 | <pext> | <starting>sort 총정리</heading> sort는 원본변경, sorted는 원본유지, 새리스트반환 val list = mutableListOf(1, 2, 7, 6, 5, 6) list.sort() println(list) // [1, 2, 5, 6, 6, 7] val list = mutableListOf(1, 2, 7, 6, 5, 6) val sorted = list.sorted() println(sorted) // [1, 2, 5, 6, 6, 7] println(list) // [1, 2, 7, 6, 5, 6] (sorted 를 사용했기 때문에 변하지 않음) // 1. sortByDescending 로 내림차순 정렬 list.sortByDescending { it } val sorted = list.sortedByDescending { it } // 2. reverse 사용해서 정렬 후 뒤집기 list.sort() list.reverse() val sorted = list.sorted().reversed() val list = mutableListOf(1 to "a", 2 to "b", 7 to "c", 6 to "d", 5 to "c", 6 to "e") list.sortBy { it.second } println(list) // [(1, a), (2, b), (7, c), (5, c), (6, d), (6, e)] val list = mutableListOf(1 to "a", 2 to "b", 7 to "c", 6 to "d", 5 to "c", 6 to "e") list.sortWith(compareBy({it.second}, {it.first})) println(list) // [(1, a), (2, b), (5, c), (7, c), (6, d), (6, e)] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 가장 간단한 생성 메서드는 naturalOrder() 입니다. 아무런 파라미터를 필요로 하지 않으며 오름차순을 기본으로 합니다. val ascComparator = naturalOrder<Long>() val complexComparator = compareBy<Pair<Int, String?>>({it.first}, {it.second}) 위 코드에서 it.first 값을 사용해 먼저 비교를 하고 값이 같은 경우에만 it.second 비교까지 이루어집니다. val caomparator = Comparator<Int> { a, b -> a.compareTo(b) } nullsFirst 또는 nullsLast 와 함께 Comparator 를 사용하면 null 값을 가장 처음 또는 가장 마지막에 위치하도록 설정할 수 있습니다. val list = mutableListOf(4, null, 1, -2, 3) list.sortWith(nullsFirst()) // [null, -2, 1, 3, 4] list.sortWith(nullsLast()) // [-2, 1, 3, 4, null] list.sortWith(nullsFirst(reverseOrder())) // [null, 4, 3, 1, -2] list.sortWith(nullsLast(compareBy { it })) // [-2, 1, 3, 4, null] 추가적인 정렬 규칙 then 키워드 val students = mutableListOf(21 to "Helen", 21 to "Tom", 20 to "Jim") val ageComparator = compareBy<Pair<Int, String?>> {it.first} val ageAndNameComparator = ageComparator.thenByDescending {it.second} // [(20, Jim), (21, Tom), (21, Helen)] println(students.sortedWith(ageAndNameComparator)) 나이가 어린 순으로 먼저 정렬하고 나이가 같으면 이름을 알파벳 역순으로 정렬합니다. </ending> | </end> |
| <hitle> | Nested Class | <chare> | 1 | <pext> | <starting>Nested Class</heading> fun main(args: Array<String>) {  val obj = Outer.Nested()  print(obj.foo()) } class Outer {  class Nested {  fun foo() = "Welcome to The TutorialsPoint.com"  } } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Welcome to The TutorialsPoint.com </ending> | </end> |
| <hitle> | class 정의 | <chare> | 1 | <pext> | <starting>class 정의</heading> class myClass {  // Property (data member)  private var name: String = "Tutorialspoint.com"  // Member function  fun printMe() {  print("The best Learning website - " + name)  } } fun main(args: Array<String>) {  val obj = myClass() // Create object obj of myClass class  obj.printMe() // Call a member function using object } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* The best Learning website - Tutorialspoint.com </ending> | </end> |
| <hitle> | Inner Class | <chare> | 1 | <pext> | <starting>Inner Class</heading> fun main(args: Array<String>) {  val obj = Outer().Inner()  print(obj.foo()) } class Outer {  private val welcomeMessage: String = "Welcome to the TutorialsPoint.com"  inner class Inner {  fun foo() = welcomeMessage  } } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Welcome to The TutorialsPoint.com </ending> | </end> |
| <hitle> | Anonymous Inner Class | <chare> | 1 | <pext> | <starting>Anonymous Inner Class</heading> fun main(args: Array<String>) {  var programmer :Human = object:Human { // Anonymous class  override fun think() { // overriding the think method  print("I am an example of Anonymous Inner Class ")  }  }  programmer.think() } interface Human {  fun think() } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* I am an example of Anonymous Inner Class </ending> | </end> |
| <hitle> | Type Aliases | <chare> | 1 | <pext> | <starting>Type Aliases</heading> typealias User = Triple<String, String, Int> fun main() {  val obj = userInfo()  print(obj) } fun userInfo():User{  return Triple("Zara","Ali",21) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* (Zara, Ali, 21) </ending> | </end> |
| <hitle> | Primary Constructor | <chare> | 1 | <pext> | <starting>Primary Constructor</heading> class Person (val \_name: String, val \_age: Int=20) {  // Member Variables  var name: String  var age: Int  // Initializer Block  init {  this.name = \_name  this.age = \_age  println("Name = $name")  println("Age = $age")  } } fun main(args: Array<String>) {  val zara = Person("Zara")  val nuha = Person("Nuha", 11) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Name = Zara Age = 20 Name = Nuha Age = 11 </ending> | </end> |
| <hitle> | Secondary Constructor | <chare> | 1 | <pext> | <starting>Secondary Constructor</heading> class Person{  // Member Variables  var name: String  var age: Int  // Initializer Block  init {  println("Initializer Block")  }  // Secondary Constructor  constructor ( \_name: String, \_age: Int) {  this.name = \_name  this.age = \_age  println("Name = $name")  println("Age = $age")  } } fun main(args: Array<String>) {  val zara = Person("Zara", 20) } //출력 Initializer Block Name = Zara Age = 20 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* class Person{  // Member Variables  var name: String  var age: Int  var salary:Double  // First Secondary Constructor  constructor ( \_name: String, \_age: Int) {  this.name = \_name  this.age = \_age  this.salary = 0.00  println("Name = $name")  println("Age = $age")  }  // Second Secondary Constructor  constructor ( \_name: String, \_age: Int, \_salary: Double) {  this.name = \_name  this.age = \_age  this.salary = \_salary  println("Name = $name")  println("Age = $age")  println("Salary = $salary")  } } fun main(args: Array<String>) {  val nuha = Person("Nuha", 12)  val zara = Person("Zara", 20, 2000.00) } //출력 Name = Nuha Age = 12 Name = Zara Age = 20 Salary = 2000.0 </ending> | </end> |
| <hitle> | class Inheritance | <chare> | 1 | <pext> | <starting>class Inheritance</heading> open class ABC {  fun think () {  println("Hey!! i am thiking ")  } } class BCD: ABC(){ // inheritence happend using default constructor  } fun main(args: Array<String>) {  var a = BCD()  a.think() } //Hey!! i am thiking  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* open class ABC {  open fun think () {  println("Hey!! i am thinking ")  } } class BCD: ABC() { // inheritance happens using default constructor   override fun think() {  println("I am from Child")  } } fun main(args: Array<String>) {  var a = BCD()  a.think() } //I am from Child  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* class BCD: ABC() {  final override fun think() {  println("I am from Child")  } } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* open class ABC {  open val count: Int = 0  open fun think () {  println("Hey!! i am thinking ")  } } class BCD: ABC() {  override val count: Int  init{  count = 100  }  override fun think() {  println("I am from Child")  }  fun displayCount(){  println("Count value is $count")  } } fun main(args: Array<String>) {  var a = BCD()  a.displayCount() } //Count value is 100 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* open class ABC {  open val count: Int = 0  open fun think () {  println("Hey!! i am thinking ")  } } class BCD(override val count: Int = 400): ABC() {  override fun think() {  println("I am from Child")  }  fun displayCount(){  println("Count value is $count")  } } fun main(args: Array<String>) {  var a = BCD(200)  var b = BCD()  a.displayCount()  b.displayCount() } //출력 Count value is 200 Count value is 400 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* open class Base {  init{  println("I am in Base class")  } } open class Child: Base() {  init{  println("I am in Child class")  } } class GrandChild: Child() {  init{  println("I am in Grand Child class")  } } fun main(args: Array<String>) {  var a = GrandChild() } //출력 I am in Base class I am in Child class I am in Grand Child class \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* open class Base() {  open val name:String  init{  name = "Base"  }  open fun displayName(){  println("I am in " + this.name)  } } class Child(): Base() {  override fun displayName(){  super.displayName()  println("I am in " + super.name)    } } fun main(args: Array<String>) {  var a = Child()  a.displayName() } //출력 I am in Base I am in Base \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* open class Rectangle {  open fun draw() { /\* ... \*/ } } interface Polygon {  fun draw() { /\* ... \*/ } // interface members are 'open' by default } class Square() : Rectangle(), Polygon {  // The compiler requires draw() to be overridden:  override fun draw() {  super<Rectangle>.draw() // call to Rectangle.draw()  super<Polygon>.draw() // call to Polygon.draw()  } } </ending> | </end> |
| <hitle> | Abstract Class | <chare> | 1 | <pext> | <starting>Abstract Class</heading> abstract class Person(\_name: String) {  var name: String  abstract var age: Int   // Initializer Block  init {  this.name = \_name  }  abstract fun setPersonAge(\_age:Int)  abstract fun getPersonAge():Int  fun getPersonName(){  println("Name = $name")  } } class Employee(\_name: String): Person(\_name) {  override var age: Int = 0  override fun setPersonAge(\_age: Int) {  age = \_age  }  override fun getPersonAge():Int {  return age  } } fun main(args: Array<String>) {  val employee = Employee("Zara")  var age : Int  employee.setPersonAge(20)  age = employee.getPersonAge()  employee.getPersonName()  println("Age = $age") } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Name = Zara Age = 20 </ending> | </end> |
| <hitle> | interface | <chare> | 1 | <pext> | <starting>interface</heading> interface ExampleInterface {  var myVar: String // abstract property  fun absMethod() // abstract method  fun sayHello() = "Hello there" // method with default implementation } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* interface ExampleInterface {  var myVar: Int // abstract property  fun absMethod():String // abstract method  fun hello() {  println("Hello there, Welcome to TutorialsPoint.Com!")  } } class InterfaceImp : ExampleInterface {  override var myVar: Int = 25  override fun absMethod() = "Happy Learning " } fun main(args: Array<String>) {  val obj = InterfaceImp()  println("My Variable Value is = ${obj.myVar}")  print("Calling hello(): ")  obj.hello()  print("Message from the Website-- ")  println(obj.absMethod()) } //출력 My Variable Value is = 25 Calling hello(): Hello there, Welcome to TutorialsPoint.Com! Message from the Website-- Happy Learning  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* interface A {  fun printMe() {  println(" method of interface A")  } } interface B {  fun printMeToo() {  println("I am another Method from interface B")  } } // implements two interfaces A and B class multipleInterfaceExample: A, B  fun main(args: Array<String>) {  val obj = multipleInterfaceExample()  obj.printMe()  obj.printMeToo() } //출력 method of interface A I am another Method from interface B </ending> | </end> |
| <hitle> | Visibility Control (Modifiers) public,private,protected,internal | <chare> | 1 | <pext> | <starting>Visibility Control (Modifiers) public,private,protected,internal</heading> class publicExample {  val i = 1  fun doSomething() {  } } //아래와 같음 public class publicExample {  public val i = 1    public fun doSomething() {  } } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* open class A() {  private val i = 1  fun doSomething(){  println("Inside doSomething" )  println("Value of i is $i" )  } } class B : A() {  fun printValue(){  doSomething()  // println("Value of i is $i" )  } } fun main(args: Array<String>) {   val a = A()  val b = B()    b.printValue() } //출력 Inside doSomething Value of i is 1 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* open class A() {  protected val i = 1  protected fun doSomething(){  println("Inside doSomething" )  println("Value of i is $i" )  } } class B : A() {  fun printValue(){  doSomething()  println("Value of i is $i" )  } } fun main(args: Array<String>) {   val a = A()  val b = B()  //a.doSomething()  b.printValue() } //출력 Inside doSomething Value of i is 1 Value of i is 1 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* package com.tutorialspoint.modifiers open class A() {  internal val i = 1  internal fun doSomething(){  println("Inside doSomething" )  println("Value of i is $i" )  } } class B : A() {  fun printValue(){  doSomething()  println("Value of i is $i" )  } } fun main(args: Array<String>) {   val a = A()  val b = B()  a.doSomething()  b.printValue() }  //출력 Inside doSomething Value of i is 1 Inside doSomething Value of i is 1 Value of i is 1 </ending> | </end> |
| <hitle> | Extension Function | <chare> | 1 | <pext> | <starting>Extension Function</heading> class Alien {  var skills : String = "null"    fun printMySkills() {  print(skills)  }  } fun main(args: Array<String>) {  var a1 = Alien()  a1.skills = "JAVA"  //a1.printMySkills()    var a2 = Alien()  a2.skills = "SQL"  //a2.printMySkills()    var a3 = Alien()  a3.skills = a1.addMySkills(a2)  a3.printMySkills() } fun Alien.addMySkills(a:Alien):String{  var a4 = Alien()  a4.skills = this.skills + " " +a.skills  return a4.skills } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* JAVA SQL </ending> | </end> |
| <hitle> | Extended Library Classes | <chare> | 1 | <pext> | <starting>Extended Library Classes</heading> fun main(args: Array<String>) {  val str = "Good morning Kotlin"  val result = str.countVowels()  println("Number of vowels: $result") } fun String.countVowels(): Int{  var vowels = 0  for (i in 0.. this.length - 1) {  val ch = this[i]  if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u') {  ++vowels  }  }  return vowels;  } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Number of vowels: 6 </ending> | </end> |
| <hitle> | Companion Object Extensions | <chare> | 1 | <pext> | <starting>Companion Object Extensions</heading> fun main(args: Array<String>) {  println("Heyyy!!!"+A.show()) } class A {  companion object {  fun show():String {  return("You are learning Kotlin from TutorialsPoint.com")  }  } } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Heyyy!!! You are learning Kotlin from TutorialsPoint.com </ending> | </end> |
| <hitle> | Extension with Nullable Receiver | <chare> | 1 | <pext> | <starting>Extension with Nullable Receiver</heading> fun main(args: Array<String>) {  var str1 = "Good morning Kotlin"  var str2 : String? = null    var result = str1.countVowels()  println("Number of vowels in str1 : $result")    result = str2.countVowels()  println("Number of vowels in str2 : $result") } fun String?.countVowels(): Any{  if (this == null) return "null"    var vowels = 0  for (i in 0.. this.length - 1) {  val ch = this[i]  if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u') {  ++vowels  }  }  return vowels;  } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Number of vowels in str1 : 6 Number of vowels in str2 : null </ending> | </end> |
| <hitle> | Extension Properties | <chare> | 1 | <pext> | <starting>sqlite</heading> class Temperature(var celsius: Float)  fun main(args: Array<String>) {  val t = Temperature(40f)  println(t.fahrenheit)   t.fahrenheit = 85f  println(t.celsius) } var Temperature.fahrenheit: Float  get() = (celsius \* 9 / 5) + 32  set(value) {  celsius = (value - 32) \* 5 / 9  } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 104.0 29.444445 </ending> | </end> |
| <hitle> | data class | <chare> | 1 | <pext> | <starting>data class</heading> data class Book(val name: String, val publisher: String, var reviewScore: Int) fun main(args: Array<String>) {  val book = Book("Kotlin", "Tutorials Point", 10)  println("Name = ${book.name}")  println("Publisher = ${book.publisher}")  println("Score = ${book.reviewScore}") } //출력 Name = Kotlin Publisher = Tutorials Point Score = 10 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* //Copy Function data class Book(val name: String, val publisher: String, var reviewScore: Int) fun main(args: Array<String>) {  val original = Book("Kotlin", "Tutorials Point", 10)  val copied = original.copy(reviewScore=5)  println("Original Book")  println("Name = ${original.name}")  println("Publisher = ${original.publisher}")  println("Score = ${original.reviewScore}")    println("Copied Book")  println("Name = ${copied.name}")  println("Publisher = ${copied.publisher}")  println("Score = ${copied.reviewScore}") } //출력 Original Book Name = Kotlin Publisher = Tutorials Point Score = 10 Copied Book Name = Kotlin Publisher = Tutorials Point Score = 5 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* //toString Function data class Book(val name: String, val publisher: String, var reviewScore: Int) fun main(args: Array<String>) {  val book = Book("Kotlin", "Tutorials Point", 10)  println(book.toString()) } //출력 Book(name=Kotlin, publisher=Tutorials Point, reviewScore=10) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* //hashCode() and equals() Functions data class Book(val name: String, val publisher: String, var reviewScore: Int) fun main(args: Array<String>) {  val original = Book("Kotlin", "Tutorials Point", 10)  val copy1 = original.copy(reviewScore=5)  val copy2 = original.copy(reviewScore=7)  println("Original Hashcode = ${original.hashCode()}")  println("Copy1 Hashcode = ${copy1.hashCode()}")  println("Copy2 Hashcode = ${copy2.hashCode()}")  if( copy1.equals(copy2)){  println("Copy1 is equal to Copy2.")  }else{  println("Copy1 is not equal to Copy2.")  } } //출력 Original Hashcode = 1957710662 Copy1 Hashcode = 1957710657 Copy2 Hashcode = 1957710659 Copy1 is not equal to Copy2. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* //Destructuring Declarations data class Book(val name: String, val publisher: String, var reviewScore: Int) fun main(args: Array<String>) {  val book = Book("Kotlin", "Tutorials Point", 10)  val( name, publisher,reviewScore ) = book  println("Name = $name")  println("Publisher = $publisher")  println("Score = $reviewScore") } //출력 Name = Kotlin Publisher = Tutorials Point Score = 10 </ending> | </end> |
| <hitle> | Sealed Class | <chare> | 1 | <pext> | <starting>Sealed Class</heading> sealed class MyExample {  class OP1 : MyExample() // MyExmaple class can be of two types only  class OP2 : MyExample() } fun main(args: Array<String>) {  val obj: MyExample = MyExample.OP2()   val output = when (obj) { // defining the object of the class depending on the inuputs   is MyExample.OP1 -> "Option One has been chosen"  is MyExample.OP2 -> "option Two has been chosen"  }  println(output) } //option Two has been chosen </ending> | </end> |
| <hitle> | Generic in/out 총정리 | <chare> | 1 | <pext> | <starting>Generic 총정리</heading> fun main(args: Array<String>) {  var objet = genericsExample<String>("JAVA")  var objet1 = genericsExample<Int>(10) } class genericsExample<T>(input:T) {  init {  println("I am getting called with the value "+input)  } } //출력 I am getting called with the value JAVA I am getting called with the value 10 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* fun <T> wrap(value: T) {  println(value) } fun main() {  wrap(1)  wrap("abc")  wrap(1.3) } //1 //abc //1.3 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* fun main() {  val intList: List<Int> = listOf(1,2,3)  val stringArray: Array<String> = arrayOf("hi", "hello") } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* open class Animal class Cat : Animal() class Dog : Animal() // 일반 상속 관계. 부모에 자식을 사용 할 수 있다. val animal: Animal = Cat() val cats: Array<Cat> = arrayOf(Cat(), Cat()) val animals: Array<Animal> = cats // Error - Type mismatch: inferred type is Array<Cat> but Array<Animal> was expected \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* fun myAnimals(animals: Array<Animal>) {  animals[0] = Dog() // Array<Cat> cats[0] = Dog() //엄청난 오유코드 } fun main() {  val cats: Array<Cat> = arrayOf(Cat(), Cat())  myAnimals(cats) } //하지만 다음코드는 문제가 없다 fun myAnimals(animals: List<Animal>) {  println(animals[0]) } fun main() {  val cats: List<Cat> = listOf(Cat(), Cat())  myAnimals(cats) } //왜냐하면 public class Array<T> public interface List<out E> \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* fun copyFromTo(from: Array<Animal>, to: Array<Animal>) {  for (i in from.indices) {  to[i] = from[i]  } } fun main() {  val animals: Array<Animal> = arrayOf(Animal(), Animal())  val cats: Array<Cat> = arrayOf(Cat(), Cat())  copyFromTo(cats,animals)// Error - Type mismatch: inferred type is Array<Cat> but Array<Animal> was expected } //out를 리용하여 우의 문제 해결 fun copyFromTo(from: Array<out Animal>, to: Array<Animal>) {  for (i in from.indices) {  to[i] = from[i]  } } fun main() {  val animals: Array<Animal> = arrayOf(Animal(), Animal())  val cats: Array<Cat> = arrayOf(Cat(), Cat())   copyFromTo(cats,animals) } //write하면 문제발생 fun copyFromTo(from: Array<out Animal>, to: Array<Animal>) {  for (i in from.indices) {  to[i] = from[i]  }  from[0] = Cat() // Error - Type mismatch: inferred type is Cat but Nothing was expected } fun main() {  val animals: Array<Animal> = arrayOf(Animal(), Animal())  val cats: Array<Cat> = arrayOf(Cat(), Cat())  copyFromTo(cats,animals) } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* fun copyFromTo(from: Array<out Animal>, to: Array<Animal>) {  for (i in from.indices) {  to[i] = from[i]  } } fun main() {  val anys: Array<Any> = arrayOf(Any(), Any())  val cats: Array<Cat> = arrayOf(Cat(), Cat())  copyFromTo(cats,anys) // Error - Type mismatch: inferred type is Array<Any> but Array<Animal> was expected } //in을 사용하여 우의 문제해결 fun copyFromTo(from: Array<out Animal>, to: Array<in Animal>) {  for (i in from.indices) {  to[i] = from[i]  } } fun main() {  val anys: Array<Any> = arrayOf(Any(), Any())  val cats: Array<Cat> = arrayOf(Cat(), Cat())    copyFromTo(cats,anys) } //write은 가능하지만 제네릭 타입으로의 read는 불가능하다. fun copyFromTo(from: Array<out Animal>, to: Array<in Animal>) {  for (i in from.indices) {  to[i] = from[i]  }  val any: Animal = to[0] //Error - Type mismatch: inferred type is Any? but Animal was expected } fun main() {  val anys: Array<Any> = arrayOf(Any(), Any())  val cats: Array<Cat> = arrayOf(Cat(), Cat())   copyFromTo(cats,anys) } </ending> | </end> |
| <hitle> | Destructuring Declarations | <chare> | 1 | <pext> | <starting>Destructuring Declarations</heading> fun main(args: Array<String>) {  val s = Student("TutorialsPoint.com","Kotlin")  val (name,subject) = s  println("You are learning "+subject+" from "+name) } data class Student( val a :String,val b: String ){  var name:String = a  var subject:String = b } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* You are learning Kotlin from TutorialsPoint.com </ending> | </end> |
| <hitle> | Exception Handling | <chare> | 1 | <pext> | <starting>Exception Handling</heading> fun main(args: Array<String>) {  try {  val myVar:Int = 12;  val v:String = "Tutorialspoint.com";  v.toInt();  } catch(e:Exception) {  e.printStackTrace();  } finally {  println("Exception Handeling in Kotlin");  } } \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* val myVar:Int = 12; Exception Handeling in Kotlin </ending> | </end> |
| <hitle> | Delegation by | <chare> | 1 | <pext> | <starting>Delegation by</heading> interface Base {  fun printMe() //abstract method } class BaseImpl(val x: Int) : Base {  override fun printMe() { println(x) } //implementation of the method } class Derived(b: Base) : Base by b // delegating the public method on the object b fun main(args: Array<String>) {  val b = BaseImpl(10)  Derived(b).printMe() // prints 10 :: accessing the printMe() method  } //10 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* class Example {  var p: String by Delegate() } class Delegate {  operator fun getValue(thisRef: Any?, property: KProperty<\*>): String {  return "$thisRef, thank you for delegating '${property.name}' to me!"  }  operator fun setValue(thisRef: Any?, property: KProperty<\*>, value: String) {  println("$value has been assigned to '${property.name} in $thisRef.'")  } } //While reading, getValue() method will be called and while setting the variable setValue() method will be called. </ending> | </end> |
| <hitle> | Property Delegation Lazy(), Observable() | <chare> | 1 | <pext> | <starting>Property Delegation Lazy(), Observable()</heading> val myVar: String by lazy {  "Hello" } fun main(args: Array<String>) {  println(myVar +" My dear friend") } //Hello My dear friend \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* import kotlin.properties.Delegates class User {  var name: String by Delegates.observable("Welcome to Tutorialspoint.com") {  prop, old, new ->  println("$old -> $new")  } } fun main(args: Array<String>) {  val user = User()  user.name = "first"  user.name = "second" } //first -> second </ending> | </end> |
| <hitle> | flatmap() vs map() | <chare> | 1 | <pext> | val groceryShopping = listOf(  ShoppingBag(listOf("Mango", "Apple", "Banana")),  ShoppingBag(listOf("LadyFinger", "Broccoli", "Onion")),  ShoppingBag(listOf("Spices", "Knife", "Plates"))  )  val clothesShopping = listOf(  ShoppingBag(listOf("Pant", "Shirt", "Tie")),  ShoppingBag(listOf("Socks", "Shoes", "Jeans")),  ShoppingBag(listOf("Jackets", "Caps", "Muffler"))  )  val grocery = groceryShopping.flatMap {  it.items  }  println(grocery)  val clothes = clothesShopping.map {  it.items  }  println(clothes)  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  [Mango, Apple, Banana, LadyFinger, Broccoli, Onion, Spices, Knife, Plates]  [[Pant, Shirt, Tie], [Socks, Shoes, Jeans], [Jackets, Caps, Muffler]] | </end> |