Creating Collections

Arrays		
Simple Array	`val intArray: Array< Int> = arrayOf(1, 2, 3)`	
Copy of Array	`val copyOf Array: Array< Int> = intArr ay.c op yOf()`	
Partial copy of Array	`val partia lCo pyO fArray: Array< Int> = intArr ay.c op yOf Ran ge(0, 2)`	
Lists		
Simple List	`val intList: List<I nt> = listOf(1, 2, 3)`	Or `arrayL ist Of( 1,2,3 )`
Empty List	`val emptyList: List<I nt> = emptyL ist()`	Or `listOf()`
List with no null elements	`val listWi thN onN ull Ele ments: List<I nt> = listOf Not Null(1, null, 3)`	same as `List(1,3)`
Sets		
Simple Set	`val aSet: Set<In t> = setOf(1)`	Or `hashSe tOf(1)` / `linked Ser Of(1)`
Empty Set	`val emptySet: Set<In t> = emptySet()`	Or `setOf()` / `hashSe tOf()` / `linked SetOf()`
Maps		
Simple Map	`val aMap: Map<St ring, Int> = mapOf( " hi" to 1, " hel lo" to 2)`	Or `mapOf( Pai r("h i", 1)` / `hashMa pOf ("hi " t o 1)` / `linked Map Of( " - hi" to 1)`
Empty Map	`val emptyMap: Map<St ring, Int> = emptyMap()`	Or `mapOf()` / `hashMa pOf()` / `linked MapOf()`
Black sheep, mutables		
Simple Mutable List	`val mutabl eList: Mutabl eLi st< Int> = mutabl eLi stOf(1, 2, 3)`	
Simple Mutable Set	`val mutabl eSet: Mutabl eSe t<I nt> = mutabl eSe tOf(1)`	
Simple Mutable Map	`var mutabl eMap: Mutabl eMa p<S tring, Int> = mutabl eMa pOf ("hi " to 1, " hel lo" to 2)`	

We will be using these collections throughout the cheat sheet.

By **Xantier**
cheatography.com/xantier/

Published 14th June, 2017.
Last updated 14th June, 2017.

Operators

Method	Example	Result	Explanation
	Iterables		
Plus	`intList + 1`	`[1, 2, 3, 1]`	Returns a new iterables with old values + added one
Plus (Iterable)	`intList + listOf(1, 2, 3)`	`[1, 2, 3, 1, 2, 3]`	Return a new iterables with old values + values from added iterable
Minus	`intList - 1`	`[2, 3]`	Returns a new iterables with old values - subtracted one
Minus (Iterable)	`intList - listOf(1, 2)`	`[3]	Returns a new iterables with old values - values from subtracted iterable
	Maps		
Plus	`aMap + Pair("H i", 2)`	`{hi=1, hello=2, Goodbye=3}`	Returns new map with old map values + new Pair. Updates value if it differs
Plus (Map)	`aMap + mapOf( Pai r("h ell o", 2), Pair("G ood by e ", 3)`	`{hi=1, hello=2, Goodbye=3}`	Returns new map with old map values + Pairs from added map. Updates values if they differ.
Minus	`aMap - Pair("H i", 2)`	{Hi=2}	Takes in a key and removes if found
Minus (Map)	aMap - listOf("hello", "hi")	`{}`	Takes in an iterable of keys and removes if found
	Mutables		
Minus Assign	`mutabl eList -= 2`	`[1, 3]`	Mutates the list, removes element if found. Returns boolean
Plus Assign	`mutabl eList += 2`	`[1, 3, 2]`	Mutates the list, adds element. Returns boolean

Operators (cont)			
Minus Assign (MutableMap)	`mutabl eMa p.m inu sAs sig n("h ell o ")`	{hi=1}	Takes in key and removes if that is found from the mutated map. Returns boolean. Same as `-=`
Plus Assign (MutableMap)	`mutabl eMa p.p lus Ass ign ("Go odb y e" to 3)`	{hi=1, Goodbye=3}	Takes in key and adds a new pair into the mutated map. Returns boolean. Same as `+=`

Transformers

Method	Example	Result	Explanation
Associate	`intLis t.a sso ciate { Pair(i t.t oSt ring(), it) }`	`{1=1, 2=2, 3=3}`	Returns a Map containing key-value pairs created by lambda
Map	`intLis t.map { it + 1 }`	`[2,3,4]`	Returns a new list by transforming all elements from the initial Iterable.
MapNotNull	`intLis t.m apN otNull { null }`	`[]`	Returned list contains only elements that return as not null from the lamdba
MapIndexed	`intLis t.m apI ndexed { idx, value -> if (idx == 0) value + 1 else value + 2 }`	`[2,4,5]`	Returns a new list by transforming all elements from the initial Iterable. Lambda receives an index as first value, element itself as second.
MapIndexe-dNotNull	`intLis t.m apI nde xed NotNull { idx, value -> if (idx == 0) null else value + 2 }`	`[4,5]`	Combination of Map, MapIndexed & MapInd-exedNotNull
MapKeys	`aMap.m apKeys { pair -> pair.key + ", mate" }`	`{hi, mate=1, hello, mate=2}`	Transforms all elements from a map. Receives a Pair to lambda, lamdba return value is the new key of original value

By **Xantier**
cheatography.com/xantier/

Published 14th June, 2017.
Last updated 14th June, 2017.

Transformers (cont)			
MapValues	`aMap.m apV alues { pair -> pair.value + 2 })`	`{hi=3, hello=4}`	Transforms all elements from a map. Receives a Pair to lambda, lamdba return value is the new value for the original key.
Reversed	`intLis t.r eve rsed())`	`[3,2,1]`	
Partition	`intLis t.p art ition { it > 2 })`	`Pair([ 1,2], [3] )`	Splits collection into to based on predicate
Slice	`intLis t.s lic e(1..2))`	`[2,3]`	Takes a range from collection based on indexes
Sorted	`intLis t.s ort ed())`	`[1,2,3]`	
SortedByD-escending	`intLis t.s ort edB yDe sce nding { it }`	`[3,2,1]`	Sorts descending based on what lambda returns. Lamdba receives the value itself.
SortedWith	`intLis t.s ort edW ith (Co mpa rat or< Int> { x, y -> when { x == 2 -> 1 y == 2 -> -1 else -> y - x } })`	`[3,1,2]`	Takes in a Comparator and uses that to sort elements in Iterable.
Flatten	`listOf (in tList, aSet).f la tten()`	`[2,3,4,1]`	Takes elements of all passed in collections and returns a collection with all those elements
FlatMap with just return	`listOf (in tList, aSet).f latMap { it }`	`[2,3,4,1]`	Used for Iterable of Iterables and Lambdas that return Iterables. Transforms elements and flattens them after transform-ation.

Transformers (cont)

FlatMap with transform	`listOf (in tList, aSet).f latMap { iterable: Iterab le< Int> -> iterable.map { it + 1 } }`	`[2,3,4,2]`	FlatMa often u with monad contain to fluen handle context errors a side effects.
Zip	`listOf(3, 4).zip (in tList)`	`[(3,1), (4,2)]`	Create list of Pairs fr two Iterable As mar pairs a values shorter the original Iterable
Zip with predicate	`listOf(3, 4).zip (in tList) { firstElem, secondElem -> Pair(firstElem - 2, secondElem + 2) }`	`[(1,3), (2,4)]`	Create list of Pairs fr two Iterable As mar pairs a values shorter the original Iterable Lambd receive both ite on that index fr Iterable
Unzip	`listOf (Pa ir( " hi", 1), Pair("h ell o", 2)).un zip()`	`Pair([hi, hello], [1,2])`	Revers the operati from `z` Takes i an Iterable Pairs a returns them a Pair of Lists.

Aggregators

Method	Example	Result	Explanation
	Folds And Reduces		
Fold	`intLis t.f old(10) { accumu lator, value -> accumu lator + value }`	`16` *(10+1+2+3)*	Accumulates values starting with initial and applying operation from left to right. Lambda receives accumulated value and current value.
FoldIndexed	`intLis t.f old Ind exe d(10) { idx, accumu lator, value -> if (idx == 2) accumu lator else accumu lator + value }`	`13` *(10+1+2)*	Accumulates values starting with initial and applying operation from left to right. Lambda receives index as the first value.

Aggregators (cont)			
FoldRight	`intLis t.f old Rig ht(10) { accumu lator, value -> accumu lator + value }`	`16` *(10+3+2+1)*	Accumulates values starting with initial and applying operation from right to left. Lambda receives accumulated value and current value.
FoldRi-ght-Indexed	`intLis t.f old Rig htI nde xed(10) { idx, accumu lator, value -> if (idx == 2) accumu lator else accumu lator + value }`	`16` *(10+3+2+1)*	
Reduce	`intLis t.r educe { accumu lator, value -> accumu lator + value }`	`6` *(1+2+3)*	Accumulates values starting with first value and applying operation from left to right. Lambda receives accumulated value and current value.
Reduce-Right	`intLis t.r edu ceRight { accumu lator, value -> accumu lator + value }`	`6` *(3+2+1)*	Accumulates values starting with first value and applying operation from right to left. Lambda receives accumulated value and current value.
Reduce-Indexed	`intLis t.r edu ceI ndexed { idx, accumu lator, value -> if (idx == 2) accumu lator else accumu lator + value }`	`3` *(1+2)*	
Reduce-RightI-ndexed	`intLis t.r edu ceR igh tIn dexed { idx, accumu lator, value -> if (idx == 2) accumu lator else accumu lator + value }`	`3` *(2+1)*	

Grouping

By **Xantier**
cheatography.com/xantier/

Published 14th June, 2017.
Last updated 14th June, 2017.

Aggregators (cont)

GroupBy

```
intLis t.g roupBy { value -> 2 }
```

GroupBy (With new values)

```
intLis t.g rou pBy({ it }, { it + 1 })
```

GroupByTo

```
val mutabl eSt rin gTo ListMap = mapOf( " fir st" to 1, " sec ond " to 2)
mutableStringToListMap.values.groupByTo(mutableMapOf<Int, Mutabl eLi st< Int >>(), { value: Int -
 })
```

GroupingBy -> FoldTo

```
intLis t.g rou pingBy { it }
.foldTo(mutableMapOf<Int, Int>(), 0) { accumu lator, element ->
          accumu lator + element     }
```

Grouping > Aggregate

```
intLis t.g rou pingBy { " key " }
.aggregate({ key, accumu lator: String?, element, isFirst ->
             when (accum ulator) {
                           null -> " $el eme nt"
                          else -> accumu lator + " $el eme nt"
                     }
     })
```


By **Xantier**
cheatography.com/xantier/

Published 14th June, 2017.
Last updated 14th June, 2017.
Page 7 of 15.

Aggregators (cont)			
Count	`intLis t.c ount()`	`3`	AKA size
Count (with Lambda)	`intLis t.count { it == 2 })`	`1`	Count of elements satisfying the predicate
Average	`intLis t.a ver age()`	`2.0` *((1+2+3)/3 = 2.0)*	Only for numeric Iterables
Max	`intLis t.max()`	`3`	Maximum value in the list. Only for Iterables of Comparables.
MaxBy	`intLis t.maxBy { it * 3 }`	`3`	Maximum value returned from lambda. Only for Lambdas returning Comparables.
MaxWith	`intLis t.m axW ith (on eOr Larger)`	`1`	Maximum value defined by passed in Comparator
Min	`intLis t.min()`	`1`	Minimum value in the list. Only for Iterables of Comparables.
MinBy	`intLis t.minBy { it * 3 }`	`1`	Minimum value returned from lambda. Only for Lambdas returning Comparables.
MinWith	`intLis t.m inW ith (on eOr Larger)`	`3`	Minimum value defined by passed in Comparator
Sum	`intLis t.sum()`	`6`	Summation of all values in Iterable. Only numeric Iterables.
SumBy	`intLis t.sumBy { if(it == 3) 6 else it } )`	`9` *(1+2+6)*	Summation of values returned by passed in lambda. Only for lambdas returning numeric values.
SumByD-ouble	`intLis t.s umB yDouble { it.toD ouble() }`	`6.0`	Summation to Double values. Lambdareceives the value and returns a Double.

```
val oneOrLarger = Comparator<Int> { x, y ->
  when{
    x == 1 -> 1
    y == 1 -> -1
    else -> y - x
  }
}
```

By **Xantier**
cheatography.com/xantier/

Published 14th June, 2017.
Last updated 14th June, 2017.

Filtering and other predicates + simple HOFs

Method	Example	Result	Notes
	Filtering		
Filter	`intLis t.f ilter { it > 2 }`	`[3]`	Filter-in
FilterKeys	`aMap.f ilt erKeys { it != " hel lo" }`	`{hi=1}`	
Filter-Values	`aMap.f ilt erV alues { it == 2 }`	`{hello=2}`	
Filter-Indexed	`intLis t.f ilt erI ndexed { idx, value -> idx == 2 || value == 2 }`	`[2,3]`	
FilterIsI-nstance	`intLis t.f ilt erI sIn sta nce <St rin g>()`	`[]`	All of them are ints
	Taking and Dropping		
Take	`intLis t.t ake(2)`	`[1,2]`	Take n elements from Iterable. If passed in number larger than list, full list is returned.
TakeWhile	`intLis t.t ake While { it < 3 }`	`[1,2]`	
TakeLast	`intLis t.t ake Last(2)`	`[2,3]`	
TakeLa-stWhile	`intLis t.t ake Las tWhile { it < 3 }`	`[]`	Last element already satisfies this condition --> empty
Drop	`intLis t.d rop(2)`	`[3]`	Drop n elements from the start of the Iterable.
DropWhile	`intLis t.d rop While { it < 3 }`	`[3]`	
DropLast	`intLis t.d rop Last(2)`	`[1]`	
DropLa-stWhile	`intLis t.d rop Las tWhile { it > 2 }`	`[1, 2]`	
	Retrieving individual elements		
Component	`intLis t.c omp one nt1()`	`1`	There are 5 of these --> `comp on ent1()`, `compon ent2( )`, `compon ent3()`, `compon - ent4()`, `compon ent5()`
ElementAt	`intLis t.e lem ent At(2)`	`3`	Retrieve element at his index. Throws IndexOutOfBounds if element index doesn't exist
ElementAt-OrElse	`intLis t.e lem ent AtO rEl se(13) { 4 }`	`4`	Retrieve element at his index or return lambda value if element index doesn't exist.

Filtering and other predicates + simple HOFs (cont)			
ElementAtOrNull	`intLis t.e lem ent AtO rNu ll(6 66)`	`null`	Retrieve element at his index or return null if element index doesn't exist.
Get (clumsy syntax)	`intLis t.g et(2)`	`3`	Get element by index
Get	`intList[2]`	`3`	Shorthand and preferred way for the one above
GetOrElse	intList.getOrElse(14) { 42 }	`42`	Get element or return lambda value if it doesn't exist.
Get from Map (clumsy syntax)	`aMap.g et( " hi")`	`1`	
Get from Map	`aMap["h i"]`	`1`	
GetValue	`aMap.getValue("hi")1	`1`	Get value or throw NoSuchElementException
GetOrDefault	`aMap.g etO rDe fau lt( " HI", 4 )`	`4`	Get value or return the value returned from lambda
GetOrPut	`mutabl eMa p.g etO rPu t("H I") { 5 }`	`5`	MutableMap only. Returns the the value if it exist, otherwise puts it and returns put value.

		Finding	
BinarySearch	`intLis t.b ina ryS ear ch(2)`	`1`	Does a binary search through the collection and returns the index of the element if found. Otherwise returns negative index.
Find	`intLis t.find { it > 1 }`	`2`	First element satisfying the condition or null if not found
FindLast	`intLis t.f indLast { it > 1 }`	`3`	Last element satisfying the condition or null if not found
First	`intLis t.f irst()`	`1`	First element of Iterable or throws NoSuchElementException
First with predicate	`intLis t.first { it > 1 }`	`2`	Same as find but throws NoSuchElementException if not found
FirstOrNull	`intLis t.f irs tOr Null()`	`1`	Throw safe version of `first()`.

By **Xantier**
cheatography.com/xantier/

Published 14th June, 2017.
Last updated 14th June, 2017.

Filtering and other predicates + simple HOFs (cont)			
FirstOrNull with predicate	`intLis t.f irs tOrNull { it > 1 }`	2	Throw safe version of `first(() -> Boolean)`.
IndexOf	`intLis t.i nde xOf(1)`	0	
IndexOfFirst	`intLis t.i nde xOf First { it > 1 }`	1	
IndexOfLast	`intLis t.i nde xOfLast { it > 1 }`	2	
Last	`intLis t.l ast()`	3	Throws NoSuchElementException if empty Iterable
Last with predicate	`intLis t.last { it > 1 }`	3	Throws NoSuchElementException if none found satisfying the condition.
LastIndexOf	`intLis t.l ast Ind exOf(2)`	1	
LastOrNull	`intLis t.l ast OrN ull()`	3	Throw safe version of `last()`
LastOrNull with predicate	`intLis t.l ast OrNull { it > 1 }`	3	Throw safe version of `last(() -> Boolean)`.
Unions, distincts, intersections etc.			
Distinct	`intLis t.d ist inct()`	`[1, 2, 3]`	
DistinctBy	`intLis t.d ist inctBy { if (it > 1) it else 2 }`	`[1,3]`	
Intersect	`intLis t.i nte rse ct( lis tOf(1, 2))`	`[1,2]`	
MinusElement	`intLis t.m inu sEl eme nt(2)`	`[1,3]`	
MinusElement with collection	`intLis t.m inu sEl eme nt( lis tOf(1, 2))`	`[3]`	
Single	`listOf ("One Elemen t").s in gle()`	`One Element`	Returns only element or throws.
SingleOrNull	`intLis t.s ing leO rNull()`	null	Throw safe version of `single()`.
OrEmpty	`intLis t.o rEm pty()`	[1, 2[, 3]	Returns itself or an empty list if itself is null.
Union	`intLis t.u nio n(l ist Of( 4,5,6))`	`[1,2,3 ,4,5,6]`	
Union (infix notation)	`intList union listOf (4,5,6)`	`[1,2,3 ,4,5,6]`	

Checks and Actions

Method	Example	Res
	Acting on list elements	
	`val listOf Fun ctions = listOf({ print( " first ") }, { print( " second ") })`	
ForEach	`listOf Fun cti ons.fo rEach { it() }`	fir
ForEachIn-dexed	`listOf Fun cti ons.fo rEa chI ndexed { idx, fn -> if (idx == 0) fn() else print( " Won't do it") }`	fir
OnEach	`intLis t.o nEach { print(it) }`	123
	Checks	
All	`intLis t.all { it < 4 }`	tru
Any	`intLis t.any()`	tru
Any with predicate	`intLis t.any { it > 4 }`	fal
Contains	`intLis t.c ont ains(3)`	tru
ContainsAll	`intLis t.c ont ain sAl l(l ist Of(2, 3, 4))`	fal
Contains (Map)	`aMap.c ont ain s("H ell o")`	fal
Contai-nsKey	`aMap.c ont ain sKe y("h ell o")`	tru
Contai-nsValue	`aMap.c ont ain sVa lue(2)`	tru
None	`intLis t.n one()`	fal
None with predicate	`intLis t.none { it > 5 }`	tru
IsEmpty	`intLis t.i sEm pty()`	fal
IsNotEmpty	`intLis t.i sNo tEm pty()`	tru

<3 Kotlin

Github repository with all code examples:
https://github.com/Xantier/Kollections

PDF of this cheat sheet:
http://jussi.hallila.com/Kollections/

Created with <3 by Jussi Hallila

By **Xantier**
cheatography.com/xantier/

Published 14th June, 2017.
Last updated 14th June, 2017.