Lists (and strings)

Concatenation	`my_list = list_1 + list_2`
Append to the end	`my_lis t.a ppe nd( new _el ement)`
Check if in list	`element in my_list`
Find in list	`position = my_lis t.i nde x(e lem ent)`
Remove last element	`my_lis t.pop()`
Insert element at position	`my_lis t.i nse rt( pos ition, ele ment)`
Remove from list (by position)	`my_lis t.p op( pos ition)`
Remove from list (by element)	`my_lis t.r emo ve( ele ment)`
Sort	`my_lis t.s ort()`
Reverse order	`my_lis t.r eve rse()`
Count elements	`number = my_lis t.c oun t(e lemen t)`

The same operations apply to strings, understood as lists of characters.

Sets

Add element	`my_set.ad d(e lement)`
Remove random element	`my_set.pop()`
Remove specific element	`my_set.re mov e(e lement)`
Add elements from other iterable	`my_set.up dat e(i ter able)`

Union, intersection, difference	`new_set = my_set.un ion (ot her _set)`
	`new_set = my_set.in ter sec tio n(o the r _set)`
	`new_set = my_set.di ffe ren ce( oth er_ s et)}`

Dictionaries

List of keys or values	`keys = my_dic t.k eys()` `values = my_dic t.v alues()`
List of tuples (key, value)	`items = my_dic t.i tems()`
Update dictionary (or join two)	`my_dic t.u pda te( {key: val ue})` `my_dic t.u pda te( oth er_ d ict)`
Remove item by key	`my_dic t.p op(key)`
Remove last added item	`my_dic t.p opi tem()`
Copy dictionary	`new_dict = my_dic t.c opy()`

List/string slicing

First/last n elements	`my_lis t[:n],my_lis t [n:]`
From element n to element m, with step s	`my_lis t[n :m:s]`

Comprehensions

List comprehension	`lc = [n ** 2 for n in range(10) if n % 2 ==`
Set comprehension	`sc = {n ** 2 for n in range(10) if n % 2 ==`
Dictionary comprehension	`dc = {n: n ** 2 for n in range(10) if n % 2`
Generator comprehension	`gc = (n ** 2 for n in range(10) if n % 2 ==`

Tuples

Count elements	`number = my_tup le.c ou nt( ele ment )`
Element in tuple	`element in my_tuple`
Find in tuple	`position = my_tup le.i nd ex( ele me nt)`

Advanced iterables

Import relevant modules	`import collec tions`
Dictionary with default values	`def_dict = collec tio ns.d ef aul td ict()`
Dictionary remembering the order of items	`ord_dict = collec tio ns.O rd ere dD ict()`

Numpy arrays

Relevant modules	`import numpy as np`

Itertools module

Import relevant modules	`import itertools`
Cycle infinitely through iterable elements	`iterto ols.cy cle (it erable)`
Loop through the cumulative sum of the iterable	`iterto ols.ac cum ula te( [1, 1,3 ,4])`
Loop through pairwise contiguous elements	`iterto ols.ac cum ula te( ite rab le)`
Cartesian product of iterables	`iterto ols.pr odu ct( iter_a, ite r_b)`
All permutations of an iterable	`iterto ols.pe rmu tat ion s(i ter able)`
Length-r combinations from an iterable	`iterto ols.co mbi nat ion s(i ter able)`

In the `accumu late()` function's argument, the iterable must contain elements which can be summed together. In other words, it can't have strings and integers at the same time as they can't be summed.