

Lists (and strings)

Concatenation	<code>my_list = list_1 + list_2</code>
Append to the end	<code>my_list.append(new_element)</code>
Check if in list	<code>element in my_list</code>
Find in list	<code>position = my_list.index(element)</code>
Remove last element	<code>my_list.pop()</code>
Insert element at position	<code>my_list.insert(position, element)</code>
Remove from list (by position)	<code>my_list.pop(position)</code>
Remove from list (by element)	<code>my_list.remove(element)</code>
Sort	<code>my_list.sort()</code>
Reverse order	<code>my_list.reverse()</code>
Count elements	<code>number = my_list.count(element)</code>

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

Sets

Add element	<code>my_set.add(element)</code>
Remove random element	<code>my_set.pop()</code>
Remove specific element	<code>my_set.remove(element)</code>
Add elements from other iterable	<code>my_set.update(iterable)</code>
Union, intersection, difference	<code>new_set = my_set.union(ot her_set)</code> <code>new_set = my_set.intersection(ot her_set)</code> <code>new_set = my_set.difference(ot her_set)</code>

Dictionaries

List of keys or values	<code>keys = my_dict.keys()</code> <code>values = my_dict.values()</code>
List of tuples (key, value)	<code>items = my_dict.items()</code>
Update dictionary (or join two)	<code>my_dict.update({key: value})</code> <code>my_dict.update(ot her_dict)</code>
Remove item by key	<code>my_dict.pop(key)</code>
Remove last added item	<code>my_dict.popitem()</code>
Copy dictionary	<code>new_dict = my_dict.copy()</code>

List/string slicing

First/last n elements	<code>my_list[:n], my_list[n:]</code>
From element n to element m, with step s	<code>my_list[n:m:s]</code>

Comprehensions

List comprehension	<code>lc = [n ** 2 for n in range(10) if n % 2 == 0]</code>
Set comprehension	<code>sc = {n ** 2 for n in range(10) if n % 2 == 0}</code>
Dictionary comprehension	<code>dc = {n: n ** 2 for n in range(10) if n % 2 == 0}</code>
Generator comprehension	<code>gc = (n ** 2 for n in range(10) if n % 2 == 0)</code>

Tuples

Count elements	<code>number = my_tuple.count(element)</code>
Element in tuple	<code>element in my_tuple</code>
Find in tuple	<code>position = my_tuple.index(element)</code>

Advanced iterables

Import relevant modules	<pre>import collections</pre>
Dictionary with default values	<pre>def_dict = collections.defaultdict()</pre>
Dictionary remembering the order of items	<pre>ord_dict = collections.OrderedDict()</pre>

Numpy arrays

Relevant modules	<pre>import numpy as np</pre>
------------------	-------------------------------

Itertools module

Import relevant modules	<pre>import itertools</pre>
Cycle infinitely through iterable elements	<pre>itertools.cycle(iterable)</pre>
Loop through the cumulative sum of the iterable	<pre>itertools.accumulate([1, 1, 3, 4])</pre>
Loop through pairwise contiguous elements	<pre>itertools.accumulate(iterable)</pre>
Cartesian product of iterables	<pre>itertools.product(iter_a, iter_b)</pre>
All permutations of an iterable	<pre>itertools.permutations(iterable)</pre>
Length-r combinations from an iterable	<pre>itertools.combinations(iterable)</pre>

In the `accumulate()` 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.

