

# Python Cheat Sheet: Complex Data Types

“A puzzle a day to learn, code, and play” → Visit [finxter.com](https://finxter.com)

|                                   | Description  | Example  |
|-----------------------------------|--|--|
| <b>List</b>                       | A container data type that stores a sequence of elements. Unlike strings, lists are mutable: modification possible.  | <pre>l = [1, 2, 2] print(len(l)) # 3</pre>   |
| Adding elements                   | Add elements to a list with (i) append, (ii) insert, or (iii) list concatenation. The append operation is very fast.   | <pre>[1, 2, 2].append(4) # [1, 2, 2, 4] [1, 2, 4].insert(2,2) # [1, 2, 2, 4] [1, 2, 2] + [4] # [1, 2, 2, 4]</pre>  |
| Removal                           | Removing an element can be slower.   | <pre>[1, 2, 2, 4].remove(1) # [2, 2, 4]</pre>  |
| Reversing                         | This reverses the order of list elements.  | <pre>[1, 2, 3].reverse() # [3, 2, 1]</pre>   |
| Sorting                           | Sorts a list. The computational complexity of sorting is linear in the no. list elements.  | <pre>[2, 4, 2].sort() # [2, 2, 4]</pre>  |
| Indexing                          | Finds the first occurrence of an element in the list & returns its index. Can be slow as the whole list is traversed.  | <pre>[2, 2, 4].index(2) # index of element 4 is "0" [2, 2, 4].index(2,1) # index of element 2 after pos 1 is "1"</pre>   |
| <b>Stack</b>                      | Python lists can be used intuitively as stacks via the two list operations append() and pop().   | <pre>stack = [3] stack.append(42) # [3, 42] stack.pop() # 42 (stack: [3]) stack.pop() # 3 (stack: [])</pre>  |
| <b>Set</b>                        | A set is an unordered collection of unique elements (“at-most-once”).  | <pre>basket = {'apple', 'eggs', 'banana', 'orange'} same = set(['apple', 'eggs', 'banana', 'orange'])</pre>  |
| <b>Dictionary</b>                 | The dictionary is a useful data structure for storing (key, value) pairs.  | <pre>calories = {'apple' : 52, 'banana' : 89, 'choco' : 546}</pre>   |
| Reading and writing elements      | Read and write elements by specifying the key within the brackets. Use the keys() and values() functions to access all keys and values of the dictionary.  | <pre>print(calories['apple'] &lt; calories['choco']) # True calories['cappu'] = 74 print(calories['banana'] &lt; calories['cappu']) # False print('apple' in calories.keys()) # True print(52 in calories.values()) # True</pre>   |
| Dictionary Looping                | You can access the (key, value) pairs of a dictionary with the items() method.   | <pre>for k, v in calories.items():     print(k) if v &gt; 500 else None # 'chocolate'</pre>  |
| <b>Membership operator</b>        | Check with the ‘in’ keyword whether the set, list, or dictionary contains an element. Set containment is faster than list containment.   | <pre>basket = {'apple', 'eggs', 'banana', 'orange'} print('eggs' in basket) # True print('mushroom' in basket) # False</pre>   |
| <b>List and Set Comprehension</b> | List comprehension is the concise Python way to create lists. Use brackets plus an expression, followed by a for clause. Close with zero or more for or if clauses.<br><br>Set comprehension is similar to list comprehension. | <pre># List comprehension l = [('Hi ' + x) for x in ['Alice', 'Bob', 'Pete']] print(l) # ['Hi Alice', 'Hi Bob', 'Hi Pete'] l2 = [x * y for x in range(3) for y in range(3) if x&gt;y] print(l2) # [0, 0, 2] # Set comprehension squares = { x**2 for x in [0,2,4] if x &lt; 4 } # {0, 4}</pre> |