# Python Data Structures

The operations you actually reach for on a list, a dictionary, and a set, with the time complexity that matters when an interviewer asks why.

## 1 LIST *An ordered, indexable sequence you will grow, shrink, or scan.*

| OPERATION | SYNTAX | TIME |
|---|---|---|
| Index | `nums[i]` | O(1) |
| Append | `nums.append(x)` | O(1) |
| Insert | `nums.insert(i, x)` | O(n) |
| Pop (end) | `nums.pop()` | O(1) |
| Pop (front) | `nums.pop(0)` | O(n) |
| Slice | `nums[1:4]` | O(k) |
| Sort | `nums.sort(key=...)` | O(n log n) |
| Membership | `x in nums` | O(n) |

insert() shifts every element after it down one slot, so it costs O(n). append() is the O(1) default; reach for insert() only when the position genuinely matters.

A grid built with [[0] * cols] * rows looks fine until you write to one cell. Every row is the same underlying list, so that write shows up in all of them. Build each row with its own list comprehension instead.

## 2 DICTIONARY *Key-based lookups, grouping, or counting, anything you index by identity instead of position.*

| OPERATION | SYNTAX | TIME |
|---|---|---|
| Set | `prices[k] = v` | O(1) |
| Get (default) | `prices.get(k, 0)` | O(1) |
| Membership | `k in prices` | O(1) |
| Keys | `prices.keys()` | O(1) |
| Values | `prices.values()` | O(1) |
| Items | `prices.items()` | O(1) |
| Counter | `Counter(nums)` | O(n) |
| defaultdict | `defaultdict(list)` | O(1) |

Reading a missing key from a defaultdict still creates it. Even a check like if d[key]: inserts the default value first. Use key in d when you only want to look, not write.

Counter's + - & | operators drop any key whose result lands at zero or below. Subtract two counters and the keys that cancelled out are gone entirely, not sitting at 0.

## 3 SET *Uniqueness or a membership check you will run more than once, when order never mattered anyway.*

| OPERATION | SYNTAX | TIME |
|---|---|---|
| Add | `bag.add(x)` | O(1) |
| Membership | `x in bag` | O(1) |
| Remove | `bag.remove(x)` | O(1) |
| Discard | `bag.discard(x)` | O(1) |
| Union | `a \| b` | O(n + m) |
| Intersection | `a & b` | O(n + m) |
| Difference | `a - b` | O(n + m) |

Checking x in nums scans the whole list, O(n). The identical check on a set is O(1). If a problem tests membership inside a loop, build the set first.

Sets only hold hashable values, so a list can never go inside one. Need a set of groups, or a dict keyed by one? Convert each group to a tuple or frozenset first.