Python Lists, Indexing, Slicing & Matrices — Q&A; (Set 13)

Q1) Use both positive and negative indexing—any repercussion?

Yes, you can mix positive and negative indexes freely; there is no special penalty beyond normal bounds checks. Caveat: be consistent for readability.

Q2) Fastest way to make a list of 1,000 identical values

For immutable values (numbers, strings, None): xs = [0] * 1000.
For mutable values (dict/list), avoid shared references: rows = [{} for _ in range(1000)] or use itertools.repeat(None, 1000) and replace as needed.

Q3) Take every other element (1st, 3rd, 5th…)

Use slicing with a step: odds = lst[0::2]. For 2nd, 4th, 6th…, use lst[1::2].

Q4) Indexing vs slicing (key differences)

Aspect	Indexing lst[i]	Slicing lst[i:j:k]
Result	Single element	New list (possibly empty)
Out of range	IndexError	Clamped to bounds (no error)
Step	N/A	Supports step; negatives like [::-1]
Assignment	lst[i] = x (one)	lst[i:j:k] = … (subsequence)

Q5) What if slice indexes are out of range?

Slicing clamps start/stop to valid bounds—no IndexError. Only invalid step (e.g., 0) or illegal extended-slice assignments raise errors.

Q6) Let a function modify the caller’s list — what to avoid?

Avoid rebinding the parameter to a new list (xs = xs + [99] or xs = sorted(xs]). Do in-place changes: xs.append(…), xs[0] = …, xs[:] = sorted(xs), xs.extend(…).

Q7) What is an “unbalanced” (jagged) matrix?

A list of lists with differing row lengths, e.g., [[1,2,3],[4,5],[6]]. Algorithms must not assume rectangular shape.

Q8) Why use a comprehension or loop for arbitrarily large matrices?

To avoid aliasing and to build distinct rows/content by index. Bad: M = [[0]*m] * n (rows alias). Good: M = [[0]*m for _ in range(n)] or M = [[i*j for j in range(m)] for i in range(n)].