Seminar 5 Lists: At the end of the document you have 40 extra exercises, choose and do!

Part 1 — List Basics

Mini-notes

- A **list** is an ordered, mutable sequence: [].
- Elements can be of mixed types.
- len(L) gives number of elements.
- Indexing starts at 0; negatives count from the end.

Guided checks (type & indexing)

```
L = [10, "hi", 3.5, True]

len(L)            # ?

type(L)           # ?

L[0], L[-1]       # ?

L[1] = "hello"      # mutate

L
```

Exercises (no loops)

1. Create A = [1, 2, 3] and B = ["a", "b"]. Make C = A + B and D = B + A. Compare C and D.
2. Start with X = []. Do X.append(5) then X.append([6, 7]). Inspect X. Explain why it's not [5, 6, 7].
3. With Y = [1, 2, 3], do Y.extend([4, 5]). Compare extend vs append on a fresh [1,2,3].

Part 2 — Slicing Deep Dive

Mini-notes

- General form: L[start:stop:step] (stop is **exclusive**).
- Defaults: start=0, stop=len(L), step=1.
- Negative indices/steps allowed.
- **Slicing returns a new list** (copy of that region), not a view.

Quick probes

S = [0,1,2,3,4,5,6,7,8,9]

S[:] # full copy

S[2:7] # 2..6

S[:4] # 0..3

S[6:] # 6..end

S[::2] # evens

S[1::2] # odds

S[::-1] # reversed copy

S[7:2:-1] # descending slice (7..3)

Slice assignment and del (structural edits, no loops)

T = [10,20,30,40,50]

T[1:3] = [99, 98, 97] # grow in-place

T[3:4] = [] # remove a chunk

del T[:2] # delete prefix

Slicing exercises (design for mastery)

1. Given L = ["a","b","c","d","e","f","g"], produce each (no loops):
    - ["b","c","d"]
    - ["a","c","e","g"]
    - ["g","f","e","d"]
    - a **reversed copy** of L
    - ["a","b","X","Y","e","f","g"] using slice assignment
2. Start with N = [1,2,3,4,5,6,7,8,9]:
    - Replace the middle three elements with [0,0,0] using **one** slice assignment (no loops).
    - Remove the last two elements using a slice (not pop).

- Keep only every third element using slicing and rebind back into N.

3. Given R = [0,1,2,3,4,5,6,7,8,9], set R to the reversed list **in-place** using slice assignment (one line).

Part 3 — Mutability, Aliasing, and Copying

Mini-notes

- list is **mutable**; strings are not.
- B = A makes **two names** pointing to the **same** list.
- To copy: A[:], list(A), or A.copy().
- **Pitfall**: grid = [[0]*3]*4 duplicates the **same inner list** 4 times.

Probes

A = [1,2]

B = A

B.append(3)

A, B # both changed (same object)

C = [1,2]

D = C[:]

D.append(3)

C, D # different objects

Exercises

1. Show with code that changing one element in B = A also changes A.
2. Make an independent shallow copy of A three different ways; mutate the copy and confirm A doesn't change.
3. Explain and demonstrate the trap:
4. grid = [[0]*3]*4
5. grid[0][0] = 1

6. # Why did multiple rows change?
7. Create a 4×3 zero “grid” **without** the aliasing trap using only literals, repetition, and slicing/copy (still no loops).

Part 4 — Concatenation & Repetition

Mini-notes

- + concatenates lists; * repeats list (shallow).
- Repetition duplicates references for nested structures (beware aliasing as above).

Exercises

1. Create [0,0,0,0,0,0] using repetition.
2. Create ["A","A","B","B","B","C"] using only concatenation and repetition.
3. Given base = [1,2,3], create [1,2,3,1,2,3,1,2,3] without loops.

Part 5 — Core List Methods

Mini-notes (in-place unless noted)

- Add/remove: append(x), extend(iterable), insert(i,x), remove(x), pop([i]), clear()
- Info: index(x), count(x)
- Order: sort(reverse=False), reverse()
- Copy: copy() (shallow)
- Built-ins: len, sum, min, max, sorted(iterable, reverse=...) returns **new** list

Exercises

1. Start with L = [3,1,4,1,5]
    - Append 9, then extend with [2,6].
    - Insert 0 at the front.
    - Remove the **first** 1 only.
    - Pop the last element; store it in last_val.

- Sort L in ascending order in place; then create L_desc as a **separate** descending copy using sorted.

2. With W = ["a","b","a","c","a","b"], compute W.count("a") and find the index of the first "b".
3. Reverse a list in place; then reverse **copy** without changing the original (use slicing).

Part 6 — Strings ↔ Lists Interop

Mini-notes

- "a,b,c".split(",") → ["a","b","c"]
- " a b c ".split() splits on any whitespace runs.
- "sep".join(list_of_strings) → combine into one string.
- list("hello") → list of characters.

Exercises

1. Split "red,green,blue,," by comma. Explain the trailing empty elements.
2. Join ["2025","10","02"] into "2025-10-02" using "-".join(...).
3. Convert a string to a list of characters and back to a string **without** loops.
4. Given "a b c", show split() (no arg) vs split(" ") results and explain the difference.

Part 7 — Capstone Tasks

(No loops; conditionals allowed.)

1. **Slice Composer**
   Given S = [0,1,2,3,4,5,6,7,8,9], build each list using **one** slicing expression or slice assignment:
   - [2,3,4,5]
   - [0,2,4,6,8]
   - [9,8,7]
   - make S become [9,8,7,6,5,4,3,2,1,0] in place

- replace the middle four elements with [99,98,97,96] using slice assignment

2. **Non-overlapping Patch**
   Start: P = [1,2,3,4,5,6,7,8].
   Replace positions 2..5 (inclusive) with ["x","y"] using **one** slice assignment; show the final P.

3. **Safe Copy Builder**
   Without loops, create a copy Q of [[1,2],[3,4]] such that changing Q[0][0] does **not** change the original. Explain the limitation of shallow copy with nested lists and provide a 2-line workaround using literal re-construction.

4. **Stable Merge Without Loops**
   Given sorted small lists A=[1,3,5], B=[2,4,6], construct [1,2,3,4,5,6] **without loops**. (Hint: interleave with slicing and concatenation in two steps: [A[0],B[0],A[1],B[1],A[2],B[2]].) Discuss why loops are better for general merging.

5. **Have you heard of list comprehension?**

   Try this:

```
square = [
  [16, 3, 2, 13],
  [5, 9, 11, 8],
  [9, 6, 7, 12],
  [4, 15, 14, 1]
]

print(square)                # whole matrix

print(square[0][1])          # element (0,1)

print([row[1] for row in square[0:2]])  # rows 0..1, column 1
print([row[1] for row in square[:2]])   # rows 0..1, column 1
print([row[0] for row in square[2:4]])  # rows 2..3, column 0
print([row[0] for row in square[2:]])   # rows 2..end, column 0
print([row[2:] for row in square[1:3]]) # rows 1..2, cols 2..end

input()
print([row[0:2] for row in square[1:3]]) # rows 1..2, cols 0..1

print([row[1] for row in square])       # all rows, column 1

print('sum', sum(square[0]))          # sum of row 0
```

Extra list of exercises collected from different sources:
got it — you want list exercises that **don't use loops** (no for/while, and let's also avoid list comprehensions). You can rely on **slicing, indexing, tuple unpacking, built-ins** (sum, min, max, sorted, any, all, len, reversed, zip, map, filter), and methods like .index(), .count(), .append(), .extend(), .insert(), .pop(), .remove(), .sort(), plus itertools if needed.
Here's a set of loop-free challenges:

No-Loop List Exercises

A. Slicing & Indexing

1. Given a = [1,2,3,4,5,6], produce [2,3,4] using only one slice.
2. Swap first and last elements of a without a loop.
3. Rotate a right by k=2 using slices only.
4. Reverse a without reversed() or loops.
5. Given a 2D list m, get the main diagonal of a 3×3 matrix **as a list** without loops.

B. Aggregations (built-ins only)

6. a = [5,1,9,2] → get sum, min, max, average with built-ins only.
7. Check if a is strictly increasing using all and slicing.
8. Count how many times x appears in a using only one method call.
9. Check if two lists are permutations of each other (same multiset) using only sorting and comparison.
10. Find the median of an odd-length list using sorted and indexing.

C. Construction & Reshaping

11. Concatenate a and b into a new list without loops.
12. Insert value x at index i without loops (use a method or slicing).
13. Remove all occurrences of x from a **without** loops or comprehensions (hint: use filter or repeated .remove with while is not allowed; prefer list(filter(...))).
14. Flatten [[1,2],[3,4],[5]] **without loops/comprehensions** (hint: sum(..., []) or itertools.chain).
15. Given a, create a new list where each element is doubled, without loops/comprehensions (hint: map).

D. Selection & Searching

16. Return the index of the first element greater than x (or -1 if none) without loops (hint: next with a generator is okay if you consider it not a loop; if that's disallowed, use min over indices that satisfy a condition via filter).
17. Given a, test if **any** element is negative without loops (any).
18. Given a, test if **all** elements are unique without loops (hint: len(a) == len(set(a))).
19. Find the second smallest **distinct** element without loops (hint: sorted(set(a))[1]).
20. Split a into evens and odds without loops/comprehensions (hint: filter twice).

E. Ordering & Windows

21. Sort strings by length without writing a loop (hint: sorted(a, key=len)).
22. Given a, create a list of pairwise sums [a0+a1, a1+a2, ...] without loops (hint: map with zip(a, a[1:])).
23. Check if a is a palindrome without loops (hint: a == a[::-1]).
24. Given a and window size k, compute the list of window sums without loops (hint: zip(*[a[i:] for i in range(k)]) plus map(sum, ...) is okay if range(k) is allowed; alternative: itertools.accumulate for O(1) per window).
25. Stable sort a by two keys: primary key1(x), secondary key2(x) without loops (hint: sorted(a, key=lambda x: (key1(x), key2(x)))).

F. 2D Lists (Matrices)

26. Transpose a matrix m (list of rows) without loops (hint: list(map(list, zip(*m)))).
27. Extract a column j as a list without loops (hint: list(map(lambda r: r[j], m))).
28. Sum each row without loops (hint: list(map(sum, m))).
29. Check if a matrix is upper triangular without loops (hint: all(m[i][j]==0 for i,j in ...) — if generators are disallowed, use all(map(...)) with itertools.product).
30. Concatenate two matrices vertically and horizontally without loops (hint: vertical: m1 + m2; horizontal: list(map(operator.add, m1, m2)) on row lists).

G. Strings ↔ Lists (still no loops)

31. Given a CSV line, split into fields and strip spaces without loops (hint: list(map(str.strip, s.split(',')))).
32. Join list of words into a sentence with commas and “and” before the last (Oxford comma) using only slices and join.
33. Count vowels in a lowercased string without loops (hint: sum(map(s.count, 'aeiou'))).
34. Remove all digits from a string without loops (hint: ''.join(filter(str.isalpha, s))).
35. Deduplicate words preserving order without loops/comprehensions (harder: use itertools.groupby on sorted indices or dict.fromkeys(s.split())).

H. Tricky/Fun (loop-free)

36. Interleave two lists [a0,a1,...] and [b0,b1,...] → [a0,b0,a1,b1,...] using sum(zip(...), ()) then list, or itertools.chain.from_iterable(zip(...)).
37. From a, take every 3rd element starting at index 1 using a single slice.
38. Remove the first and last 2 elements using slices only.
39. Build a list of cumulative sums without loops (hint: itertools.accumulate).
40. Evaluate a polynomial at x with coefficients c0..cn without loops (hint: functools.reduce for Horner’s method).