FREE EDITION · NOTES + 3 SAMPLE MCQS

CUET · COMPUTER SCIENCE · CLASS XI · CODE 308

Lists

CUET unit: Lists (Python Data Structures)

By **UniDrill** · NCERT-grounded study material

WWW.UNIDRILL.IN

📌 Snapshot

- A Python **list** is an ordered, mutable sequence that can hold elements of different data types in a single structure.
- The foundational vocabulary (indexing, slicing, traversal) underpins all Python data-structure work tested at CUET UG level.
- CUET tests list methods, slicing output prediction, mutability/aliasing behaviour, and the difference between operations such as `append()` vs `extend()` and `sort()` vs `sorted()`.
- Eight areas matter (Introduction, Operations, Traversal, Methods/Built-ins, Nested Lists, Copying, List as Argument, List Manipulation), each of which has appeared in NTA sample/past papers.
- Numerical output-prediction questions on lists are the most common MCQ format in the Computer Science paper.

📖 Detailed Notes

2.1 Core concepts

- **List definition and syntax:** A list is an ordered, mutable sequence enclosed in square brackets with elements separated by commas. Unlike a string (characters only), a list can hold integers, floats, strings, tuples, or even other lists. List indices start from 0. (NCERT §9.1, p. 189)
- **Accessing elements (positive indexing):** `list1[0]` returns the first element; `list1[3]` returns the fourth. Accessing an index beyond the length raises `IndexError`. An expression resulting in an integer (e.g., `list1[1+4]`) is a valid index. (NCERT §9.1.1, p. 190)
- **Negative indexing:** Index `-1` returns the last element; `-n` (where n = `len(list)`) returns the first. `list1[n-1]` and `list1[-1]` both reach the last element. (NCERT §9.1.1, p. 190)
- **Mutability:** Lists are mutable — an element can be overwritten by direct assignment, e.g., `list1[3] = 'Black'` changes the fourth element in place. Strings do not allow this. (NCERT §9.1.2, p. 190)
- **Concatenation (`+`):** Joins two or more lists into a new list; the original lists remain unchanged. Both operands must be of type list; concatenating a list with a string raises `TypeError`. (NCERT §9.2.1, p. 191)

- **Repetition (`*`):** Replicates a list a given number of times, e.g., `['Hello'] * 4` produces a list of four `'Hello'` strings. (NCERT §9.2.2, p. 191)
- **Membership (`in` / `not in`):** `in` returns `True` if the element is present; `not in` returns `True` if it is absent. Both work exactly as they do for strings. (NCERT §9.2.3, p. 191)
- **Slicing:** Extracts a sub-list using `list[start:stop:step]`. If `start > stop` and step is positive, an empty list is returned. A negative step reverses the list. Out-of-range stop index is truncated to the end of the list. (NCERT §9.2.4, p. 192)
- **Traversal with `for` loop:** `for item in list1: print(item)` iterates over each element directly. Alternatively, `for i in range(len(list1)): print(list1[i])` uses indices. (NCERT §9.3, p. 192–193)
- **Traversal with `while` loop:** Use a counter `i = 0` incremented inside the loop; loop continues while `i < len(list1)`. (NCERT §9.3, p. 193)
- **List methods — modifying:** `append(x)` adds a single element (or a list as one element) at the end; `extend(list2)` adds each element of `list2` individually; `insert(i, x)` places `x` at index `i`; `remove(x)` deletes the first occurrence of `x` (raises `ValueError` if absent); `pop(i)` removes and returns element at index `i` (last element if no argument); `reverse()` reverses in place; `sort()` sorts in place (accepts `reverse=True`). (NCERT §9.4, Table 9.1, pp. 193–195)
- **List methods — querying:** `len(list1)` returns total number of elements; `count(x)` returns how many times `x` appears; `index(x)` returns the index of the first occurrence (raises `ValueError` if absent); `min()`, `max()`, `sum()` return the minimum, maximum, and sum of elements respectively. (NCERT §9.4, Table 9.1, pp. 193–195)
- **`sorted()` vs `sort()`:** `sorted(list1)` creates a **new** sorted list while leaving `list1` unchanged; `list1.sort()` modifies `list1` in place. (NCERT §9.4, Table 9.1, p. 195)
- **Nested lists:** A list that appears as an element of another list is a nested list. Access requires two indices: `list1[i][j]` where `i` selects the nested list and `j` selects the element within it. (NCERT §9.5, p. 195)
- **Copying lists — aliasing trap:** `list2 = list1` does NOT create a new list; both variables reference the same object (aliasing). Changes to one are reflected in the other. (NCERT §9.6, p. 196)
- **Three methods for a true (independent) copy:** (1) Slice: `list2 = list1[:]`; (2) Built-in: `list2 = list(list1)`; (3) Library: `import copy; list2 = copy.copy(list1)`. (NCERT §9.6, pp. 196–197)
- **List as function argument — pass by reference:** When a list is passed to a function, the function receives a reference to the same object. Any in-place changes (e.g., modifying elements via index) are reflected in the caller's list. (NCERT §9.7, pp. 197–198)

- **List as function argument — new assignment:** If the parameter is assigned a completely new list inside the function (`list2 = [...]`), Python creates a new local object; the caller's list is unaffected. The `id()` function confirms the change in reference. (NCERT §9.7, pp. 198–199)
- **List manipulation programs:** A menu-driven program (Program 9-3) covers append, insert, extend, modify, delete by position/value, sort ascending/descending, and display. Programs 9-4 and 9-5 illustrate computing average marks and linear search using lists passed to functions. (NCERT §9.8, pp. 199–203)

2.2 Definitions to memorise

Term	Definition	Page
List	An ordered, mutable sequence of elements enclosed in square brackets and separated by commas; elements can be of different data types	189
Nested list	A list that is itself an element of another list; accessed using two indices `list[i][j]`	195
Aliasing	When two variable names refer to the same list object; changes through either name affect the shared object	196
Concatenation (`+`)	An operation that joins two lists end-to-end, producing a new list without modifying the originals	191
Repetition (`*`)	An operation that replicates a list a specified number of times, producing a new (longer) list	191
Slicing	Extracting a portion of a list using `list[start:stop:step]` syntax	192
`append()`	List method that adds a single element (which may itself be a list) at the end of the list	194
`extend()`	List method that appends each element of the argument list individually to the end of the calling list	194
`pop()`	List method that removes and returns the element at the given index; removes the last element if no index is specified	194
`sort()`	In-place list method that arranges elements in ascending order (or descending if `reverse=True`)	195
`sorted()`	Built-in function that returns a new sorted list from the argument list, leaving the original unchanged	195
Pass by reference (list)	When a list is passed to a function, the function receives a reference to the original list object, so in-place modifications are visible in the caller	197–198
`insert(i, x)`	List method that places element `x` at index `i` , shifting subsequent elements right	194

Term	Definition	Page
`remove(x)`	List method that deletes first occurrence of `x`; raises ValueError if absent	194
`reverse()`	List method that reverses the list in place	195
`count(x)`	List method returning the number of times `x` occurs	195
`index(x)`	List method returning the index of the first occurrence of `x`; raises ValueError if not found	195
`len(list1)`	Built-in returning total number of elements	193
`min(list1)`	Built-in returning the smallest element	195
`max(list1)`	Built-in returning the largest element	195
`sum(list1)`	Built-in returning the sum of all numeric elements	195
Shallow copy	A new list referencing the same nested objects as the original; produced by `list1[:]`, `list(list1)`, `copy.copy()`	196-197
Mutability	Property of lists allowing in-place modification of elements	190
`copy.copy()`	Function in the `copy` module that returns a shallow copy of an object	197

2.3 Diagrams / processes to remember

- **Two-way indexing diagram** (p. 190): For `list1 = [2,4,6,8,10,12]`, positive indices run 0–5 left to right; negative indices run -6 to -1 right to left. Useful for negative-index output questions.
- **Table 9.1 — Built-in functions for list manipulations** (pp. 193–195): A three-column table (Method | Description | Example) covering `len()`, `list()`, `append()`, `extend()`, `insert()`, `count()`, `index()`, `remove()`, `pop()`, `reverse()`, `sort()`, `sorted()`, `min()`, `max()`, `sum()`. Students should be able to predict output for each method's example.
- **Aliasing vs independent copy** (p. 196): Mental model — `list2 = list1` makes them point to the same box; `list2 = list1[:]` draws a new identical box. The `id()` values confirm this.
- **Function argument reference diagram** (Program 9-1 output, p. 198): The `id` of the list is identical inside the function and in main, confirming same-object reference. Compare with Program 9-2 where a new assignment changes the `id` inside the function.

2.4 Common confusions / NTA trap points

- `append()` **vs** `extend()` **:** `list1.append([50,60])` adds `[50,60]` as a single nested element, making the list one element longer; `list1.extend([50,60])` adds 50 and 60 as two separate elements. A very common MCQ trap.

- `sort()` **vs** `sorted()`**:** `sort()` modifies the original list and returns `None`; `sorted()` returns a new sorted list and leaves the original intact. NTA frequently asks about the state of the original list after each operation.
- **Slicing edge cases:** If `start >= stop` with a positive step, the result is an empty list `[]`, not an error. If the stop index exceeds the list length, Python truncates to the end — no `IndexError`.
- **Aliasing trap:** `list2 = list1` does NOT copy the list. After `list1.append(10)`, `list2` also shows the new element. Students mistake this for an independent copy.
- `remove()` **removes the first occurrence only (NCERT Table 9.1, p. 194).** If an element appears multiple times, only the first is deleted. Calling `remove()` on a non-existent value raises `ValueError`, not silently doing nothing.
- `pop()` **returns the removed value (NCERT Table 9.1, p. 194).** Unlike `remove()`, `pop()` returns the removed element. NTA distractor: claims `pop()` returns None.
- `sort()` **returns None, not the sorted list (NCERT Table 9.1, p. 195).** `result = list1.sort()` makes `result` be `None`. NTA traps assume `sort` returns the sorted list.
- **Lists can be heterogeneous (NCERT §9.1, p. 189).** A single list may mix ints, strings, lists, etc. NTA distractor: claims lists must contain a single data type.
- `list[i][j]` **for nested lists (NCERT §9.5, p. 195).** First index selects the sub-list; second index selects the element. NTA may swap the order.
- **Slice never raises IndexError (NCERT §9.2.4, p. 192).** Slicing tolerates out-of-range indices. NTA distractor may say `list[0:100]` raises an error.
- `list1 + list2` **returns a NEW list (NCERT §9.2.1, p. 191).** It does not modify `list1` or `list2`. NTA may suggest concatenation modifies the first list.

🎯 Practice MCQs

Q1. What will be the output of the following Python statement? ```python list1 = [10, 20, 30, 40, 50] list1.append([60, 70]) print(len(list1)) ```

A. 7

B. 5

C. 6

D. Error — cannot append a list to a list

Q2. Consider the following code: ```python list1 = [1, 2, 3] list2 = list1 list1.append(4) print(list2) ``` Which of the following is the correct output?

A. `[1, 2, 3]`

B. `[1, 2, 3, 4]`

C. `[4]`

D. Error

Q3. Which of the following statements about `sort()` and `sorted()` is CORRECT?

A. Both `sort()` and `sorted()` return a new sorted list without modifying the original.

B. `sort()` modifies the list in-place and returns `None`; `sorted()` returns a new sorted list and leaves the original unchanged.

C. `sorted()` modifies the list in-place and returns `None`; `sort()` returns a new sorted list.

D. Both modify the list in-place but `sorted()` also returns the modified list.

📊 PYQ Alignment

Chapter 9 (Lists) is one of the highest-weightage topics in the CUET Computer Science paper, with questions appearing consistently across 2023–2025 sample papers; NTA particularly favours output-prediction questions on slicing, the `append()` / `extend()` distinction, aliasing behaviour, and the in-place vs new-list difference between `sort()` and `sorted()`. Statement-based and match-the-column questions on list methods (Table 9.1) and the behaviour of lists passed to functions also recur regularly. See PYQ archive for Computer Science.