FREE EDITION · NOTES + 3 SAMPLE MCQS

CUET · COMPUTER SCIENCE · CLASS XI · CODE 308

Working with Lists and Dictionaries

CUET unit: Working with Lists and Dictionaries

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

WWW.UNIDRILL.IN

📌 Snapshot

- Two fundamental Python data structures appear here — lists (mutable ordered sequences) and dictionaries (mutable key-value mappings) — and they underpin most real-world Python programs.
- CUET tests list indexing, slicing, built-in methods (append, extend, insert, pop, remove, sort, sorted), and the distinction between sort() and sorted().
- The dictionary section covers creation, access by key, membership testing, and methods such as keys(), values(), items(), get(), update(), and del.
- Both data structures are mutable, a property frequently used as a distractor in NTA questions that contrast lists with strings or tuples.
- List and dictionary traversal using for loops is a common basis for output-prediction questions in CUET.

📖 Detailed Notes

2.1 Core concepts

Section 4.1 — Introduction to List

- A list is an ordered, mutable sequence that can hold elements of different data types (integers, floats, strings, other lists). Elements are enclosed in square brackets and separated by commas. (NCERT §4.1, p. 55)
- A list containing another list as an element is called a nested list. Example: `list4 = [['Physics',101],['Chemistry',202],['Mathematics',303]]`. (NCERT §4.1, p. 56)

Section 4.1.1 — Accessing Elements in a List

- Each element is accessed using its index. The first index is 0, incrementing by 1. Negative indexing starts from the last element at index -1. (NCERT §4.1.1, p. 56)
- Accessing an index beyond the valid range raises `IndexError: list index out of range`. (NCERT §4.1.1, p. 56)
- `len(list)` returns the total number of elements; `list[n-1]` accesses the last element using the length. (NCERT §4.1.1, p. 56)

Section 4.1.2 — Lists are Mutable

- Unlike strings, a list can be modified after creation by assigning a new value to any index position: `list1[3] = 'Black'`. (NCERT §4.1.2, p. 57)

Section 4.2 — List Operations

- **Concatenation (+):** Joins two or more lists into a new list. Both operands must be list type; mixing list with string raises `TypeError`. (NCERT §4.2.1, p. 57)
- **Repetition (*):** Replicates the contents of a list a specified number of times: `list1 * 4`. (NCERT §4.2.2, p. 58)
- **Membership (in / not in):** `in` returns True if the element is present; `not in` returns True if the element is absent. (NCERT §4.2.3, p. 58)
- **Slicing `[start:stop:step]`:** Extracts a sub-list. If start > stop, an empty list is returned. A negative step (`[::-1]`) reverses the list. When the second index is out of range, slicing truncates to the end of the list. (NCERT §4.2.4, pp. 58–59)

Section 4.3 — Traversing a List

- A list can be traversed element-by-element using `for item in list` or by index using `for i in range(len(list))`. (NCERT §4.3, p. 59)
- `len(list)` returns the length; `range(n)` generates numbers from 0 to n-1. (NCERT §4.3, p. 59)

Section 4.4 — List Methods and Built-in Functions

- `len()` — returns number of elements. `list()` — creates a list from a sequence or creates an empty list. (NCERT §4.4, Table 4.1, p. 60)
- `append(x)` — adds a single element x at the end; can also append a list as a single element. `extend(list2)` — appends each element of list2 individually to the end of the original list. (NCERT §4.4, Table 4.1, p. 60)
- `insert(i, x)` — inserts element x at index i. (NCERT §4.4, Table 4.1, p. 60)
- `count(x)` — returns the number of times x appears in the list. (NCERT §4.4, Table 4.1, p. 61)
- `index(x)` (referred to as find() in table heading) — returns the index of the first occurrence of x; raises `ValueError` if absent. (NCERT §4.4, Table 4.1, p. 61)
- `remove(x)` — removes the first occurrence of x; raises `ValueError` if not present. (NCERT §4.4, Table 4.1, p. 61)
- `pop(i)` — removes and returns the element at index i; with no argument, removes and returns the last element. (NCERT §4.4, Table 4.1, p. 61)
- `reverse()` — reverses the list in place. (NCERT §4.4, Table 4.1, p. 61)
- `sort()` — sorts the list in place in ascending order; `sort(reverse=True)` sorts in descending order. (NCERT §4.4, Table 4.1, p. 61)

- `sorted()` — takes a list and returns a new sorted list without modifying the original. (NCERT §4.4, Table 4.1, p. 62)
- `min()`, `max()`, `sum()` — return the minimum, maximum, and sum of numeric list elements. (NCERT §4.4, Table 4.1, p. 62)

Section 4.5 — List Manipulation

- Practical programs demonstrate all list operations via a menu-driven interface (Program 4-1), computing averages (Program 4-2), and searching for an element by value (Program 4-3). (NCERT §4.5, pp. 62–66)

Section 4.6 — Introduction to Dictionaries

- A dictionary is a mapping data type consisting of key-value pairs (called items). Keys are separated from values by a colon (:); items are enclosed in curly braces and separated by commas. (NCERT §4.6, p. 67)
- Keys must be unique and of an immutable data type (number, string, or tuple). Values can be repeated and of any data type. (NCERT §4.6.1, p. 67)
- Items are accessed via keys rather than by position: `dict3['Ram']` returns 89. Using a key not in the dictionary raises `KeyError`. (NCERT §4.6.2, p. 68)
- The membership operator `in` checks key presence; `not in` checks key absence. (NCERT §4.6.3, p. 68)
- Dictionaries are mutable: new items are added by assigning to a new key (`dict1['Meena'] = 78`); existing values are modified by overwriting the key-value pair. (NCERT §4.6.4, pp. 68–69)

Section 4.7 — Traversing a Dictionary

- A dictionary can be traversed with `for key in dict` (Method 1) or `for key, value in dict.items()` (Method 2). (NCERT §4.7, p. 69)

Section 4.8 — Dictionary Methods and Built-in Functions

- `len()` — number of key-value pairs. `dict()` — creates a dictionary from a sequence of key-value pairs. (NCERT §4.8, Table 4.2, p. 69–70)
- `keys()` — returns all keys. `values()` — returns all values. `items()` — returns all (key, value) tuples. (NCERT §4.8, Table 4.2, p. 70)
- `get(key)` — returns the value for the key; returns None if the key is absent (unlike direct access which raises KeyError). (NCERT §4.8, Table 4.2, p. 70)
- `update(dict2)` — merges dict2's key-value pairs into the original dictionary. (NCERT §4.8, Table 4.2, p. 70)
- `clear()` — removes all items, leaving an empty dictionary `{}`. (NCERT §4.8, Table 4.2, p. 70)

- `del dict[key]` — removes the item with the specified key; `del dict_name` removes the entire dictionary from memory (raises NameError on subsequent access). (NCERT §4.8, Table 4.2, p. 71)

Section 4.9 — Manipulating Dictionaries

- Practical programs demonstrate dictionary creation, traversal, key/value/item retrieval, membership check, and character-frequency counting (Program 4-5) and digit-to-word conversion (Program 4-6). (NCERT §4.9, pp. 71–74)

2.2 Definitions to memorise

Term	Definition	Page
List	An ordered, mutable sequence of elements of possibly different data types, enclosed in square brackets.	55
Nested List	A list that contains another list as one of its elements.	56
Index	An integer value used to access an element in a list; starts at 0 for the first element and -1 for the last.	56
Concatenation	Joining two or more lists using the + operator to produce a new list; original lists remain unchanged.	57
Repetition	Replicating a list's contents using the * operator a specified number of times.	58
Slicing	Extracting a portion of a list using `list[start:stop:step]` notation.	58
append()	A list method that adds a single element at the end of the list.	60
extend()	A list method that adds each element of an iterable individually to the end of the list.	60
pop()	A list method that removes and returns the element at the given index; removes and returns the last element if no index is given.	61
sort()	A list method that sorts the list in place (modifies original); accepts `reverse=True` for descending order.	61
sorted()	A built-in function that returns a new sorted list without modifying the original list.	62
Dictionary	A mutable mapping data type consisting of unique key-value pairs enclosed in curly braces; keys must be of immutable type.	67
Item	A key-value pair in a dictionary, with key and value separated by a colon.	67
KeyError	Error raised when a key used to access a dictionary value is not present in the dictionary.	68
get()	A dictionary method that returns the value for a key, or None if the key is absent (no exception raised).	70

Term	Definition	Page
update()	A dictionary method that merges another dictionary's key-value pairs into the calling dictionary.	70
`insert(i, x)`	List method placing element x at position i, shifting subsequent elements right	60
`remove(x)`	Deletes first occurrence of x; raises ValueError if x is missing	61
`count(x)`	Number of times x appears in the list	61
`index(x)`	Index of the first occurrence of x; raises ValueError if missing	61
`clear()`	Dictionary method that removes all items leaving an empty dict object	70
`del` keyword	Deletes a specific key (`del d['k']`) or the whole dictionary (`del d`)	71
`keys()`	View of all keys in the dictionary	70
`values()`	View of all values	70
`items()`	View of all (key, value) tuples	70
Mutability	Property allowing in-place modification of elements (lists and dicts qualify; strings/tuples do not)	57
Aliasing	Two variables referencing the same list object so changes are visible through both	57
Reverse with `[::-1]`	Slicing idiom that returns a reversed copy of a list	59

2.3 Diagrams / processes to remember

- **Two-way indexing in a list** (p. 56): 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. `list1[-1]` equals `list1[5]` equals 12.
- **Slicing with step** (p. 59): `list1[0:6:2]` on a 7-element list picks elements at indices 0, 2, 4; `list1[::-1]` reverses the entire list.
- **append() vs extend() diagram** (p. 60): `append([50,60])` adds the entire list `[50,60]` as one element (list of lists); `extend([40,50])` adds 40 and 50 as two separate elements.
- **sort() vs sorted() comparison** (p. 62): `sort()` modifies the list in place; `sorted()` creates a new list and the original remains unchanged — a classic NTA trap illustrated with `list1 = [23,45,11,67,85,56]`.
- **Dictionary traversal — two methods** (p. 69): Method 1 iterates over keys and looks up values; Method 2 uses `.items()` to unpack key-value pairs directly in the for-loop header.

2.4 Common confusions / NTA trap points

- **append() vs extend():** `append([50,60])` results in `[..., [50,60]]` (nested); `extend([50,60])` results in `[..., 50, 60]` (flat). NTA frequently presents both as options for a given output.
- **sort() vs sorted():** `sort()` returns None and modifies the list in place; `sorted()` returns a new list and leaves the original intact. Printing the original after `sorted(list1)` without assignment still shows the unsorted list — a common NTA trap (Exercise Q1b, p. 75).
- **IndexError vs KeyError:** Out-of-range list index raises `IndexError`; accessing a dictionary with an absent key raises `KeyError`. Using `get()` instead avoids KeyError (returns None).
- **List mutability vs string immutability:** Lists can be changed after creation via index assignment; strings cannot. NTA likes assertion-reason questions pairing these two facts.
- **in operator on dictionaries checks keys only (NCERT §4.6.3, p. 68).** `'Suhel' in dict1` checks whether 'Suhel' is a key, not a value. Students often confuse this with checking values.
- **`del d` vs `d.clear()` (NCERT §4.8, p. 71).** `del d` removes the variable name; `d.clear()` keeps the name but empties the dict.
- **`pop()` returns the removed element (NCERT §4.4, p. 61).** It is the only deletion method that gives back the removed value; `remove()` returns None.
- **Dictionary keys must be immutable (NCERT §4.6.1, p. 67).** Lists cannot be keys; tuples and strings can.
- **Slicing never raises IndexError (NCERT §4.2.4, p. 59).** Out-of-range slice indices truncate silently.
- **Iterating a dict gives keys (NCERT §4.7, p. 69).** `for x in d:` yields keys, not values.
- **`update()` overwrites existing keys (NCERT §4.8, p. 70).** Common keys in the merged dict overwrite the originals.

🎯 Practice MCQs

Q1. What will be the output of the following Python statements? ``` list1 = [2, 4, 6, 8, 10, 12] print(list1[-1]) ```

A. 2

B. 10

C. 12

D. IndexError

Q2. Consider the following code: ``` list1 = [10, 20, 30] list2 = [40, 50] list1.append(list2) print(list1) ``` Which of the following is the correct output?

A. `[10, 20, 30, 40, 50]`

B. `[10, 20, 30, [40, 50]]`

C. `[[10, 20, 30], [40, 50]]`

D. `TypeError`

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

A. Both sort the list in place and return None.

B. `sort()` returns a new sorted list; `sorted()` modifies the list in place.

C. `sort()` modifies the list in place and returns None; `sorted()` returns a new sorted list without modifying the original.

D. Both return a new sorted list without modifying the original.

📊 PYQ Alignment

Chapter 4 on Lists and Dictionaries is among the most heavily tested topics in CUET Computer Science, typically contributing 4–6 questions per year; questions focus on output prediction from list slicing and method calls, distinguishing append() from

extend(), sort() from sorted(), and dictionary key-access versus get() behaviour — all areas where the NCERT examples directly map to past NTA question patterns. See PYQ archive for Computer Science.