PREMIUM EDITION

B.PHARMA · IST YEAR · 2026-27

Data Structures & File Handling

UNIT-III · BASICS OF PYTHON IN PHARMACEUTICAL SCIENCES

Lists, tuples, dicts, strings, NumPy and CSV - with batch, formulary and patient-dataset programs for pharmacy.

Core Concepts · Worked Examples · Infographics · Case Study · Smart Revision · Exam Points

Inside Unit III

3.1 Lists - index, slice, ops

3.2 Tuples + dicts - records

3.3 Strings - clean drug names

3.4 NumPy - dose arithmetic

3.5 CSV - patients file I/O

Levels: Remember to Apply

Pills and Python · PCI-aligned · Remember to Apply

§ UNIT OVERVIEW

3 Unit-3 at a Glance

3.1 Lists

- Ordered, *mutable* batches
- Index `w[0]` + slice `w[1:4]`
- Pharma: tablet weights

3.1 List ops

- `append` / `remove` / `sort` / `len`
- Restock program
- Stock-level lists

3.2 Tuples

- Ordered, *immutable*
- Fixed batch record
- `(id, drug, mg)` sealed

3.2 Dicts

- *key : value* pairs
- Patient record + formulary
- `get / update / keys` safe

3.3 + 3.4 NumPy

- `np.array` dose vectors
- One line beats a loop
- *mean / sum / max*

3.5 CSV files

- `csv.writer` saves rows
- `DictReader` reads back
- patients.csv mini-set

Learning Roadmap



§ 3.1 LISTS - CREATE + INDEX

1 Lists Hold Batches

Definition

A *list* is an ordered, mutable collection in `[ ]`. Each item gets a position number (index) starting at *0*; negative index `-1` means last item. In pharmacy a list holds one batch of tablet weights, stock counts or readings.

Index map - five tablet weights (mg)

Positions: *0* 1 2 3 *4* left to right; from the right: `-5 ... -1`. *w[0]* is first, *w[-1]* is last. Off-by-one is the classic viva trap - index 5 does not exist here.

batch_weights.py

```
# Batch weights in mg - synthetic teaching data
weights = [480, 495, 505, 512, 498]
print(weights[0]) # first tablet
print(weights[-1]) # last tablet
print(len(weights)) # batch size
```

Output 480 | 498 | 5

Professor's Note

Say it aloud: first is zero, last is minus-one. Every IndexError in the exam hall comes from forgetting this one sentence. Synthetic teaching data.

§ 3.1 INDEXING + SLICING

Slicing cuts a sub-batch in one move

The slice rule w[start : stop : step]

start is included, *stop* is excluded, *step* defaults to 1. So `w[1:4]` takes positions 1, 2, 3 – three items, never the 4th. `w[::-1]` reverses the whole batch.

slice_batch.py

```
# Same batch: [480, 495, 505, 512, 498]
print(weights[1:4]) # middle three
print(weights[:2]) # first two
print(weights[::-1]) # reversed QC check
```

Output [495, 505, 512] | [480, 495] | [498, 512, 505, 495, 480]

Slice	Positions taken	Result
w[1:4]	1, 2, 3	[495, 505, 512]
w[:2]	0, 1	[480, 495]
w[::-1]	all, backwards	[498, 512, 505, 495, 480]

Remember

Read every slice as start-in, stop-out. If the exam asks why w[1:4] has only three items, that sentence is the full mark. Synthetic teaching data.

§ 3.1 LIST OPERATIONS

Grow, shrink, sort and count the list

Four verbs that run a dispensary list

`append(x)` adds one item at the end +1 · `remove(x)` deletes by value −1 · `sort()` orders in place · `len()` counts. append vs extend : append adds one box, extend pours in many.

restock.py

```
# Restock program - synthetic teaching data
stock = [480, 495, 505]
stock.append(507) ; print(stock)
stock.remove(495) ; print(stock)
print(sorted([505, 480, 512]))
```

Output [480, 495, 505, 507] | [480, 505, 507] | [480, 505, 512]

Exam Tip

`remove()` deletes by *value*, `pop()` by *position* – and remove of a missing value raises `ValueError`. Examiners swap them deliberately. Synthetic teaching data.

§ 3.2 TUPLES - SEALED RECORDS

2 Tuples Seal the Record

Definition

A *tuple* is an ordered, immutable collection in `( )`. Once created it *cannot change* - any edit attempt raises `TypeError`.

Why immutable? The batch-record argument

A batch record `(id, drug, mg)` must survive audits *untouched*. Immutability is a tamper seal: the QC stamp, assay value and expiry cannot be silently overwritten the way a list can. Use tuples for facts, lists for work-in-progress.

batch_record.py

```
# Sealed batch record - synthetic teaching data
batch = ('B-1042', 'Paracetamol', 500, 25.0)
print(batch[1], len(batch))
batch[1] = 'Aspirin' # forbidden!
```

Output Paracetamol 4 | TypeError: 'tuple' object does not support item assignment

Professor's Note

One-line viva answer: lists change, tuples testify. If the question says audit, log or batch ID - write a tuple. Synthetic teaching data.

§ 3.2 DICTIONARIES

3 Dicts Name Every Field

Definition

A *dict* stores key : value pairs in { }. Look up by *name*, not by position - rec["drug"] reads itself, unlike rec[2].

patient_dict.py

```
# Patient record + mini formulary - synthetic teaching data
rec = {'name': 'Asha', 'age': 34, 'drug': 'Metformin', 'dose_mg': 500}
print(rec['drug'], rec['dose_mg'])
formulary = {'PCM': 500, 'AMX': 250}
print(formulary['PCM'])
```

Output Metformin 500 | 500

Remember

Model a *record* as a dict, a *queue of records* as a list of dicts. That single sentence describes half of all hospital datasets. Synthetic teaching data.

§ 3.2 DICT OPERATIONS

get, update and keys - the safe toolkit

Three methods, one warning

`get(k, default)` reads *without crashing* on missing keys · `update({...})` adds or overwrites fields · `keys() / values() / items()` list the contents. Direct rec[missing] raises KeyError - in a dispensary, always prefer `get`.

dict_ops.py

```
# rec from previous page
print(rec.get('drug'))
print(rec.get('allergy', 'NKDA'))
rec.update({'stock': 120})
print(list(rec.keys()))
```

Output Metformin | NKDA | ['name', 'age', 'drug', 'dose_mg', 'stock']

KeyError warning

Typing `rec["allergy"]` when the key is absent *crashes the program* with KeyError. Counselling software must never crash mid-queue - use `rec.get("allergy", "NKDA")` and the patient simply gets flagged for history-taking.

§ 3.3 STRING CLEANING

4 Strings Clean Drug Names

Definition

String methods return new, cleaned strings - the original is never changed. Dispensary feeds arrive messy (" paracetamol 500MG "), and four calls standardise them.

Method	What it does	Example result
strip()	trims edge spaces	' TAB ' -> 'TAB'
split(',')	cuts on commas	'a,500,cap' -> 3 parts
join()	glues with '-'	'Amoxicillin-500mg'
upper()/title()	case standard	'PARACETAMOL'

clean_names.py

```
print(' paracetamol 500MG '.strip().title())
print('amoxi,500,cap'.split(','))
print('-'.join(['Amoxicillin', '500mg']))
```

Output Paracetamol 500Mg | ['amoxi', '500', 'cap'] | Amoxicillin-500mg

Professor's Note

Chain them: `s.strip().title()` in one breath. Viva favourite - strings are immutable, so forgetting to capture the result means the cleaning never happened.

§ 3.4 NUMPY ARRAYS

NumPy Arrays

Definition

NumPy gives the `ndarray` – a fast, same-type number vector. One array holds 500 dose values; one operation transforms all of them with no visible loop.

arrays_create.py

```
import numpy as np
doses = np.array([250, 500, 750])
print(doses, doses.mean(), doses.sum())
print(np.zeros(3).tolist(), np.arange(0, 10, 2).tolist())
```

Output [250 500 750] 500.0 1500 | [0.0, 0.0, 0.0] [0, 2, 4, 6, 8]

Creation verbs to memorise

`np.array([...])` from a list · `np.zeros(3)` / `np.ones(4)` placeholders · `np.arange(0,10,2)` steps like range · `.mean()` / `.sum()` / `.max()` summarise instantly. All outputs above are *real executed results*.

Remember

NumPy arrays hold one type, fixed length – unlike lists. That rigidity is exactly what buys the speed for lakh-row datasets.

§ 3.4 VECTORISED DOSE CALC

One line beats a loop – dose for five weights

Vectorisation in one picture

Loop version repeats the multiply *five times* in Python. Vectorised version writes `w * 10` once and NumPy multiplies all five in compiled C – same answer, a fraction of the time on large wards.

dose_vector.py

```
import numpy as np
w = np.array([40, 50, 60, 70, 80])
doses = w * 10 # mg-per-kg rate
print(doses.tolist(), doses.mean(), doses.sum())
loop = [x * 10 for x in [40, 50, 60, 70, 80]]
print(loop)
```

Output [400, 500, 600, 700, 800] 600.0 3000 | [400, 500, 600, 700, 800]

Professor's Note

Both answers match – that is the point to write first. Then add: vectorised scales, loops crawl on 10,000 patients. Dose math verified: 60 kg x 10 mg/kg = 600 mg. Synthetic teaching data.

§ 3.5 CSV - WRITING

6 Writing patients.csv

Definition

A *CSV file* is plain text with one row per line, commas between fields. Python's `csv` module writes it; `newline=''` stops blank lines on Windows; `writerows` saves the header plus all rows.

write_patients.py

```
import csv
rows = [['pid','name','drug','dose_mg','stock'],
    ['P01','Asha','Metformin',500,120],
    ['P02','Ravi','Amoxicillin',250,8],
    ['P03','Meena','Paracetamol',500,45],
    ['P04','Kiran','Insulin',100,5]]
with open('patients.csv', 'w', newline='') as f:
    csv.writer(f).writerows(rows)
```

Output patients.csv saved - 5 lines: header + 4 patients (P01-P04)

Remember

Never hand-format CSV with commas and quotes yourself - `csv.writer` escapes names like "D'Souza, Jr." correctly. Synthetic teaching data.

§ 3.5 CSV - READING

Reading back with DictReader - real file I/O

DictReader names every column

`csv.DictReader(f)` turns each row into a *dict keyed by the header* - `row["drug"]` instead of `row[2]`. Note: every value arrives as `text`, so wrap stock in `int()` before comparing.

read_patients.py

```
import csv
with open('patients.csv') as f:
    recs = list(csv.DictReader(f))
print(len(recs), [r['name'] for r in recs])
print([r['name'] for r in recs if int(r['stock']) < 10])
```

Output 4 ['Asha', 'Ravi', 'Meena', 'Kiran'] | ['Ravi', 'Kiran']

Professor's Note

These are *real outputs from a real file* written and read by the csv module - header + 4 rows, low-stock pair Ravi and Kiran. Synthetic teaching data.

§ 3.5 DATASET ANATOMY

Columns are fields, rows are patients

How to read any healthcare table

A **column** is one field measured the same way for everyone (drug, dose_mg). A **row** is one patient visit. The header row names the columns; everything below is data. Shape of our file: **4 rows x 5 columns**.

pid	name	drug	dose_mg	stock
P01	Asha	Metformin	500	120
P02	Ravi	Amoxicillin	250	8
P03	Meena	Paracetamol	500	45
P04	Kiran	Insulin	100	5

Import + manipulate in two verbs

`import` : `DictReader` loads the file into a list of dicts · `manipulate` : filter `int(stock) < 10`, sort by dose, count by drug. Master those two and every small pharma dataset obeys you.

Remember

Synthetic teaching data. Four patients only - big enough to practise filter, sort and count, small enough to verify by eye.

§ 3.6 MASTER DEMO

Formulary lookup + low-stock report

What the demo proves

A **formulary dict** answers dose lookups in O(1); a **list-of-dicts scan** flags every stock below 10. Lookup + filter + report – the three moves of dispensary software in twelve lines.

master_demo.py

```
# formulary lookup + low-stock report - synthetic teaching data
form = {'Metformin': 500, 'Amoxicillin': 250, 'Insulin': 100}
print('Dose:', form.get('Insulin', 'unknown'))
low = [r['name'] for r in recs if int(r['stock']) < 10]
print('Reorder:', ', '.join(low))
```

Output Dose: 100 | Reorder: Ravi, Kiran

Professor's Note

Trace it before you run it: `get` never crashes on a new drug, the filter keeps only stock below 10, `join` prints one clean reorder line. Examiners award marks per line traced. Synthetic teaching data.

§ 3.7 TWO CLASSIC ERRORS

IndexError vs KeyError – never confuse them

Error	Trigger	Safe habit
IndexError	w[5] on 5-item list	check len() first
KeyError	rec["allergy"] missing	use get(k, default)
TypeError	edit a tuple	rebuild, don't assign

safe_access.py

```
weights = [480, 495, 505, 512, 498]
try: weights[5]
except IndexError: print('Bad index: only 0-4 exist')
try: rec['allergy']
except KeyError: print('Use get(): NKDA assumed')
```

Output Bad index: only 0-4 exist | Use get(): NKDA assumed

Exam trap

The question gives `w[5]` on a five-item list and asks what prints. Answer: ***nothing prints – IndexError stops the program*** unless caught. Same logic for a missing dict key and KeyError.

§ 3.8 LIST vs TUPLE vs DICT

The chooser table – one glance, full marks

Feature	list	tuple	dict
Brackets	[ ]	( )	{ k: v }
Ordered?	yes	yes	insertion order
Mutable?	yes	NO – sealed	yes (by key)
Access by	index w[0]	index t[0]	key rec["drug"]
Pharma use	batch weights	batch record	patient file

Chooser rule in one line

Changing pile? list · *Sealed fact?* tuple · *Named fields?* dict. Audit = tuple, queue = list, record = dict – write that sentence and the 5-marker is yours.

Professor's Note

Examiners love: "give one pharma example of each" – answer weights-list, batch-tuple, patient-dict and stop. No extra ink needed.

§ DEEP-DIVE EXAMPLES x3 - EXECUTED

Three drills examiners repeat

Drill 1 - negative-index slicing drill

neg_slice.py

```
# tail drills - synthetic teaching data
w = [480, 495, 505, 512, 498]
print(w[-3:]) # last three
print(w[:-1]) # drop last
print(w[-4:-1]) # middle three
```

Output [505, 512, 498] | [480, 495, 505, 512] | [495, 505, 512]

Drill 2 - dict .get() with default vs KeyError

dict_get.py

```
# safe lookup - synthetic teaching data
rec = {'drug': 'Metformin', 'dose_mg': 500}
print(rec.get('drug')) # safe hit
print(rec.get('allergy', 'NKDA')) # safe miss
print(rec['allergy']) # crashes: KeyError
```

Output Metformin | NKDA | KeyError: 'allergy'

Drill 3 - batch totals: comprehension vs loop

batch_totals.py

```
# double the stock - synthetic teaching data
stocks = [120, 8, 45, 5]
loop = [] ; for s in stocks: loop.append(s * 2)
comp = [s * 2 for s in stocks]
print(loop, comp, loop == comp)
```

Output [240, 16, 90, 10] | [240, 16, 90, 10] | True

Trace first

Drill 1: `-3:` takes the tail - Drill 2: `get` never crashes - Drill 3: same list, `one line beats four`. Synthetic teaching data.

§ EXAM HIGH-YIELD POINTS

7 Exam Points

- **Indexing:** first is `w[0]`, last is `w[-1]` – off-by-one costs the mark.
- **Slicing:** `w[1:4]` takes 1,2,3 – stop excluded, never the 4th.
- **Tuple:** immutable – batch record; edit raises `TypeError`.
- **Dict:** `get(k, default)` avoids `KeyError`; direct `rec[k]` crashes.
- **Strings:** methods return new strings – `strip / split / join / upper`; capture the result.
- **NumPy:** `w * 10` vectorises five doses at once; `.mean() / .sum()` summarise.
- **CSV:** `writerows` saves, `DictReader` reads dicts; values arrive as text – `int()` first.
- **Program:** formulary lookup + `stock < 10` filter + `join` reorder line – practise tracing it.

§ REVISE REVISION + CO3

8 Revision Grid

Concept	One-line recall	Code verb
List	ordered + mutable pile	append/remove/sort
Tuple	sealed audit fact	(id, drug, mg)
Dict	named-field record	get/update/keys
String	clean, never in place	strip/split/join
NumPy	one op, whole vector	array, *, mean
CSV	text rows, dicts back	writer, DictReader

CO3 mapping - where this unit is examined

CO3: apply data structures and file handling to store, compute and persist pharma data. Maps to Remember (recall verbs), Understand (slice/get rules) and Apply (dose vector + CSV report). The master demo is the Apply evidence.

Professor's Note

Revise backwards from the master demo: can you re-derive `get`, the slice rule and `w * 10` without looking? If yes, CO3 is done.

§ CHECK SELF-CHECK + SUMMARY

Five questions – answer before you turn the page

Q1. weights has 5 items – why does weights[5] raise IndexError?

Q2. What does weights[1:4] return, and why only three items?

Q3. Batch record must survive audit – tuple or list, and why?

Q4. rec['allergy'] is missing – which call avoids KeyError?

Q5. How does w * 10 in NumPy differ from a five-round loop?

Answers (cover, then recall)

A1. Valid indexes are 0–4; 5 is one past the end.

A2. [495, 505, 512] – stop index 4 is excluded.

A3. Tuple – immutable, tamper-sealed for audit.

A4. rec.get('allergy', 'NKDA') – default, no crash.

A5. Same values; vectorised runs once in C, loop repeats in Python.

Unit III in one line

Unit 3 teaches you how to store pharmacy data (`list / tuple / dict`), compute it (`NumPy`) and keep it (`CSV`) – so batch, dose and stock work runs itself.

§ PINPOINTS MUST REMEMBER

9 Pinpoints – Must Remember

Hook	Recall in one line
0-based index	first item is w[0]; 5 items live at 0–4
Negative index	w[-1] is last; w[-3:] takes the tail
Slice stop-out	w[1:4] takes 1,2,3 – stop excluded
get() default	rec.get(k, default) never raises KeyError
Tuple immutable	sealed batch record; edit raises TypeError
append vs extend	append adds one box, extend pours many
len() guard	check len() before w[i]; beats IndexError
CSV reader rows	DictReader gives dicts; values are text – int() first
NumPy vectorize	w * 10 hits all doses at once in C
Loop vs comp	[s*2 for s in stocks] equals the 4-line loop

3 exam predictions

1. Trace `w[-3:]` and `w[1:4]` on a 5-item batch. 2. Missing key: `get` vs `rec[k]` – which crashes? 3. Rewrite the dose vector `w * 10` plus low-stock filter `stock < 10` from memory.

Mnemonic – ZERO GET SEAL

Zero-based, tail with minus-one; GET averts KeyError, SEAL the tuple, vectorise the rest. Say ZERO-GET-SEAL once and all ten hooks follow.