Noteskarts

Smart Notes. Bright Future.

COMPUTER APPLICATIONS IN PHARMACY

Unit 3 — Data Structures & File Handling

Short Notes | Diagrams | Tables | Quick Concepts

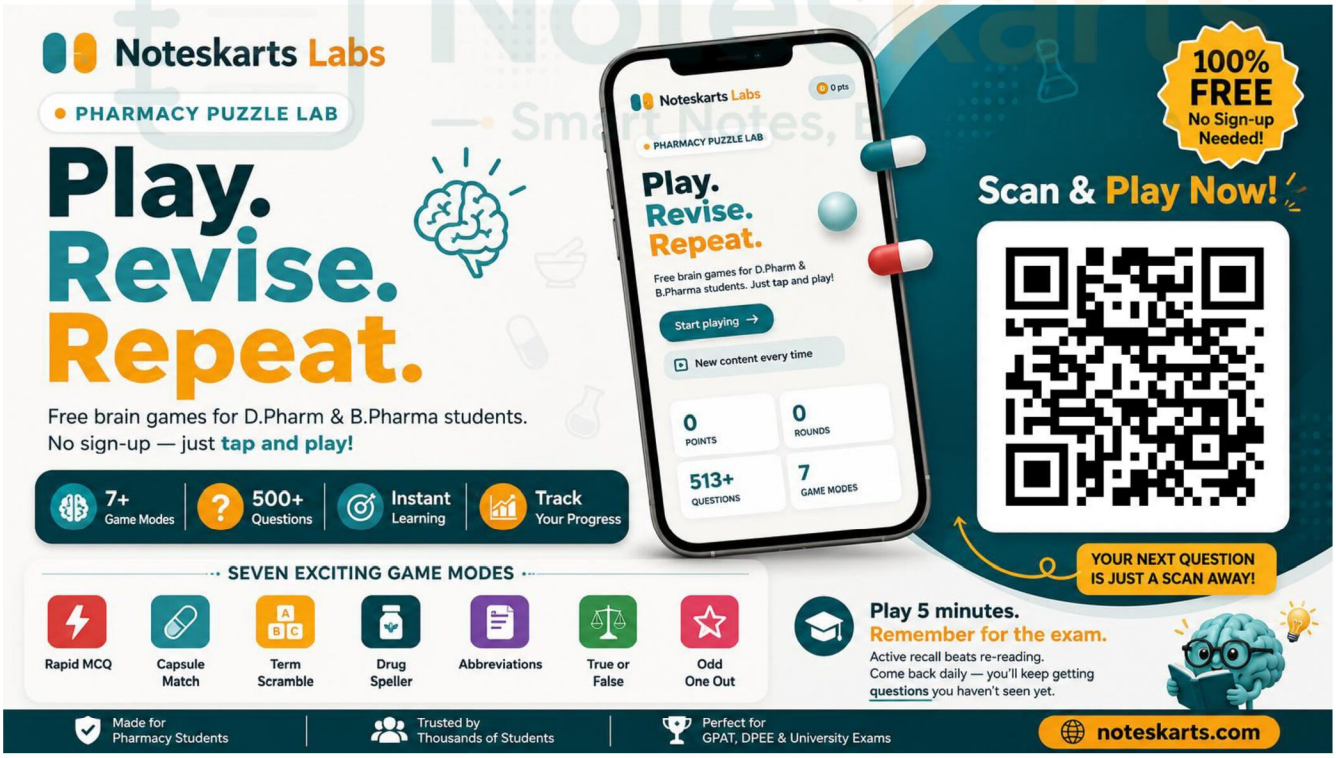
For D.Pharm & B.Pharm Students

www.noteskarts.com

Get Noteskarts Prime — Full Access

Scan the QR code above with your phone camera to download Noteskarts Prime and unlock complete detailed notes, PYQ papers, MCQ tests & video lectures for D.Pharm, B.Pharm, GPAT & Pharmacy Competitive Exams.

Free Bonus — Noteskarts Puzzle Lab



Revise this exact topic through fun MCQ games — Rapid MCQ, Term Scramble, True or False & more. 100% free, no sign-up needed. Scan and play!

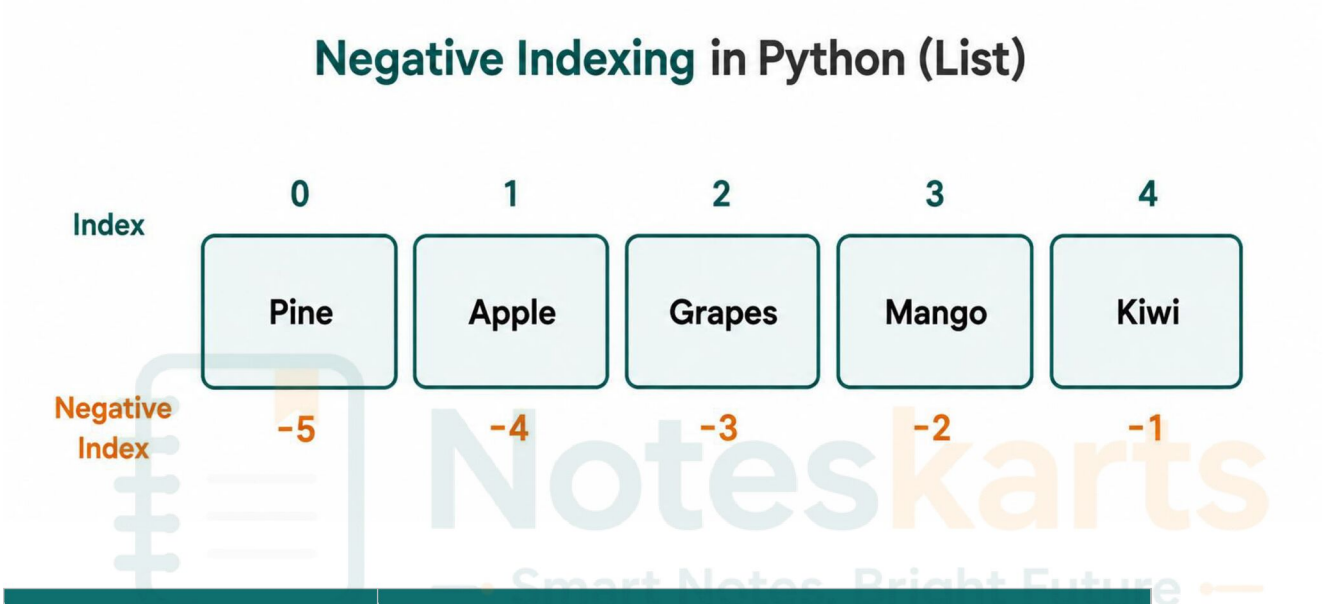
Lists - Creating, Indexing & Slicing

A list is an ordered, changeable (mutable) collection that can hold multiple items - even of different data types - in a single variable. Lists are written using square brackets [].

```
drugs = ['Paracetamol', 'Ibuprofen', 'Aspirin', 'Cetirizine',
'Dextromethorphan']
prices = [12.5, 8.0, 5.5, 15.0]
mixed = ['Tab', 500, True]     # list can hold mixed data types
```

Indexing & Slicing

Indexing accesses a single item; slicing extracts a sub-list using [start:end] (end is excluded).



Expression	Result
`drugs[0]`	'Paracetamol' (first item)
`drugs[-1]`	'Dextromethorphan' (last item)
`drugs[1:4]`	['Ibuprofen','Aspirin','Cetirizine']
`drugs[:3]`	first 3 items
`drugs[::-1]`	reversed list

More Slicing Patterns

Expression	Meaning
`drugs[2:]`	from index 2 to the end
`drugs[:-2]`	all except the last 2 items
`drugs[::2]`	every 2nd item (step = 2)
`drugs[1:4:2]`	index 1 to 3, step 2

Nested Lists (List of Lists)

A list can contain other lists - useful for representing simple tables, e.g. one row per patient.

```
patients = [
    ['Ravi', 32, 500],
    ['Sana', 45, 250],
]
print(patients[0])       # ['Ravi', 32, 500]
print(patients[0][0])    # 'Ravi'   - first item of first row
print(patients[1][2])    # 250      - dose of second patient
```

⚠ Common Mistake

drugs[5] on a 5-item list (indices 0-4) raises an IndexError - always remember the last valid index is len(list) - 1.

Basic Operations on Lists

Operation	Example	Effect
Add item	drugs.append('Metformin')	Adds to the end
Insert at position	drugs.insert(1, 'Amoxicillin')	Inserts at given index
Remove by value	drugs.remove('Aspirin')	Removes first match
Remove by index	drugs.pop(0)	Removes & returns item at index
Sort	prices.sort()	Sorts list in place (ascending)
Length	len(drugs)	Number of items in list
Check membership	'Aspirin' in drugs	Returns True / False
Combine lists	list1 + list2	Concatenates two lists

```
stock = [50, 30, 80, 15]
stock.append(60)
stock.sort()
print(stock)   # [15, 30, 50, 60, 80]
print(sum(stock), max(stock), min(stock))
```

More List Methods

Method	Example	Effect
`extend()`	list1.extend(list2)	Adds all items of list2 to list1
`count()`	stock.count(50)	Counts occurrences of a value
`reverse()`	stock.reverse()	Reverses the list in place
`index()`	drugs.index('Aspirin')	Returns position of first match
`clear()`	stock.clear()	Empties the list
`copy()`	new_list = stock.copy()	Creates an independent copy

□ Common Mistake - Copying Lists

new_list = old_list does NOT create a copy - both names point to the SAME list in memory, so changing one changes the other! Use old_list.copy() or list(old_list) for a real, independent copy.

List Comprehension (Compact List Creation)

A list comprehension builds a new list in a single line, often replacing a longer for-loop.

```
doses = [250, 500, 750, 1000]

# Traditional way
doubled = []
for d in doses:
    doubled.append(d * 2)

# List comprehension - same result, one line
doubled = [d * 2 for d in doses]

# With a condition
high_doses = [d for d in doses if d >= 500]
print(high_doses)   # [500, 750, 1000]
```

Tuples

A tuple is an ordered collection like a list, but it is immutable - once created, its values cannot be changed. Tuples use round brackets ().

```
vitals = (120, 80)          # blood pressure: systolic, diastolic
```

```
systolic, diastolic = vitals   # tuple unpacking
print('Systolic:', systolic)
```

□ Why Use a Tuple?

Tuples are used for data that should NOT change - e.g. fixed coordinates, RGB colour values, or a patient's date of birth - and they are slightly faster than lists.

Tuple Methods & When to Use

Method / Feature	Example	Note
`count()`	vitals.count(80)	Counts occurrences of a value
`index()`	vitals.index(80)	Position of first match
`Single item tuple`	t = (5,)	Comma is required, else it's just an int!

Feature	List	Tuple
`Brackets`	[ ]	( )
`Mutable?`	Yes - can add/remove/change	No - fixed after creation
`Speed`	Slower	Faster
`Typical use`	Changing collections (stock list)	Fixed records (BP reading, DOB)

Worked Example - List of Tuples for Repeated Readings

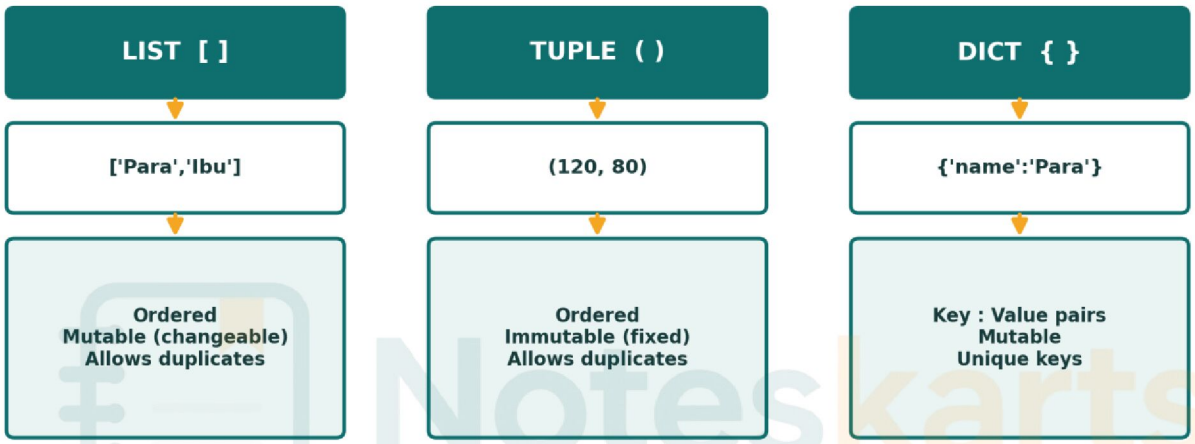
A list of tuples is a common way to store repeated fixed-structure readings, such as blood pressure taken across several visits.

```
bp_visits = [(120, 80), (128, 84), (135, 88)]
for systolic, diastolic in bp_visits:
    print(f'{systolic}/{diastolic} mmHg')
```

Dictionaries

A dictionary stores data as key:value pairs, written inside curly braces {}. Each key must be unique and is used to quickly look up its value - like a real dictionary or a drug index card.

```
drug_info = {
    'name': 'Paracetamol',
    'dose_mg': 500,
    'category': 'Analgesic'
}
print(drug_info['name'])          # access by key -> Paracetamol
drug_info['stock'] = 120          # add a new key:value pair
drug_info['dose_mg'] = 650        # update an existing value
```



Method	Purpose
`dict.keys()`	Returns all keys
`dict.values()`	Returns all values
`dict.items()`	Returns key-value pairs
`dict.get('key')`	Safely fetches a value (no error if missing)

Looping Through a Dictionary

```
stock = {'Paracetamol': 120, 'Ibuprofen': 45, 'Aspirin': 12}
for name, qty in stock.items():
    print(name, '->', qty, 'units')

# Removing a key safely
stock.pop('Aspirin')
# Updating multiple values at once
stock.update({'Ibuprofen': 60, 'Paracetamol': 150})
```

Nested Dictionary (Dict of Dicts)

Useful for storing multiple records, each identified by a unique key - like a small patient database.

```
patients = {
    'P1': {'name': 'Ravi', 'age': 32, 'dose': 500},
    'P2': {'name': 'Sana', 'age': 45, 'dose': 250},
}
print(patients['P1']['name'])   # 'Ravi'
```

Dictionary Comprehension

```
prices = {'Paracetamol': 12.5, 'Ibuprofen': 8.0, 'Aspirin': 5.5}

# Add 10% tax to every price, in one line
with_tax = {name: round(p * 1.1, 2) for name, p in prices.items()}
print(with_tax)
# {'Paracetamol': 13.75, 'Ibuprofen': 8.8, 'Aspirin': 6.05}
```

String Manipulation Techniques

Beyond basic operations, strings support several techniques useful for cleaning and formatting pharmacy/health text data.

Technique	Example	Result
`f-strings (formatting)`	f'Dose: {dose}mg'	Inserts variable values into text
`join()`	','.join(['A','B','C'])	'A,B,C'
`startswith() / endswith()`	'Tab123'.startswith('Tab')	True
`isdigit()`	'500'.isdigit()	True
`title()`	'paracetamol'.title()	'Paracetamol'
`index()`	'Aspirin'.index('p')	1

```
patient = 'ravi kumar'
dose = 500
print(f'{patient.title()} was given {dose} mg')
# Output: Ravi Kumar was given 500 mg
```

More Useful String Techniques

Technique	Example	Result
`splitlines()`	'a\nb\nc'.splitlines()	['a','b','c']
`.format()`	'{} mg'.format(500)	'500 mg'
`zfill()`	'7'.zfill(3)	'007'
`count()`	'Paracetamol'.count('a')	3
`lstrip()/rstrip()`	' drug'.lstrip()	'drug'

Common Mistake

Strings are immutable in Python - word[0] = 'X' raises a TypeError. Always create a NEW string instead, e.g. word = 'X' + word[1:].

Worked Example - Processing a Clinical Note

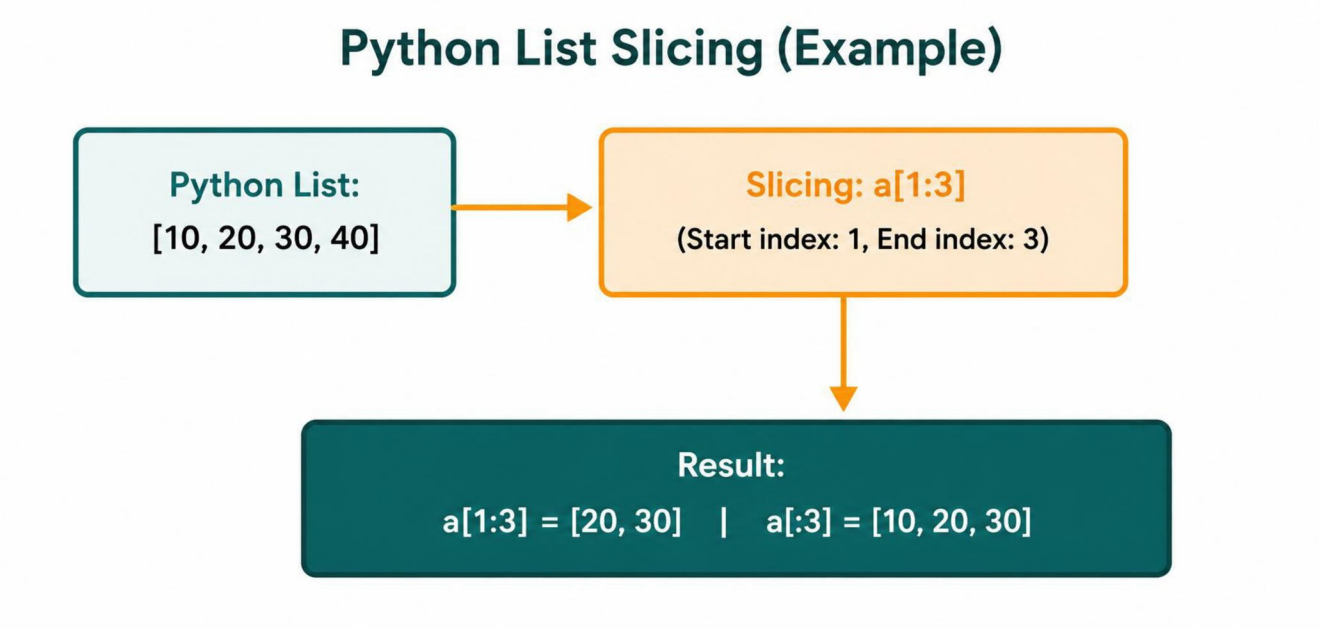
```
note = 'Patient reports mild headache and nausea after dose'
words = note.split()              # split into a list of words
print('Word count:', len(words))
print('Contains nausea:', 'nausea' in note)
cleaned = note.replace('mild', 'moderate')
print(cleaned)
```

Introduction to NumPy Arrays

NumPy (Numerical Python) is a third-party library for fast, memory-efficient numerical computing. Its core object is the ndarray (n-dimensional array), which is much faster than a plain Python list for maths on large datasets.

```
import numpy as np

doses = np.array([250, 500, 750, 1000])   # 1D array from a list
print(doses)
print(doses.shape)     # (4,)  -> 4 elements
print(doses.dtype)     # int64
```



□ List vs NumPy Array

A plain list needs a loop to add 5 to every element. A NumPy array lets you write arr + 5 directly - this is called vectorized / element-wise operation and is much faster.

More Ways to Create Arrays

Function	Creates
`np.zeros(5)`	Array of 5 zeros: [0. 0. 0. 0. 0.]
`np.ones(3)`	Array of 3 ones: [1. 1. 1.]
`np.arange(0,10,2)`	[0 2 4 6 8]
`np.linspace(0,1,5)`	5 evenly spaced values from 0 to 1
`np.array([[1,2],[3,4]])`	2D array (matrix) - 2 rows, 2 columns

```
doses_2d =
np.array([[250,500],[300,600],[200,400]])
print(doses_2d.shape)   # (3, 2) -> 3
rows, 2 columns
print(doses_2d[1, 0])   # 300 -> row 1,
column 0
```

Basic Operations Using NumPy

Function / Operation	Purpose	Example
`np.array()`	Create array from a list	np.array([1,2,3])
`np.zeros(n)`	Array of n zeros	np.zeros(4)
`np.arange(a,b)`	Array like range()	np.arange(0,10,2)
`arr + / - / * / /`	Element-wise arithmetic	arr * 2
`arr.mean()`	Average of elements	doses.mean()
`arr.sum()`	Total of elements	doses.sum()
`arr.max() / .min()`	Largest / smallest value	doses.max()

```
import numpy as np
weights =
np.array([50, 60,
70, 55])
dose_per_kg = 5

total_doses =
weights *
dose_per_kg   #
element-wise
multiply
print(total_doses)
# [250 300 350 275]
print('Average
dose:',
total_doses.mean())
```


Indexing, Slicing & Comparison

```
doses = np.array([250, 500, 750, 1000, 1250])
print(doses[1:4])          # [500 750 1000] - slicing works like lists
print(doses > 600)         # [False False True True True]
print(doses[doses > 600])  # [750 1000 1250] - boolean filtering!
```

□ Boolean Filtering

doses[doses > 600] is called BOOLEAN INDEXING - it returns only the elements where the condition is True. This is a very common technique for filtering health/lab data.

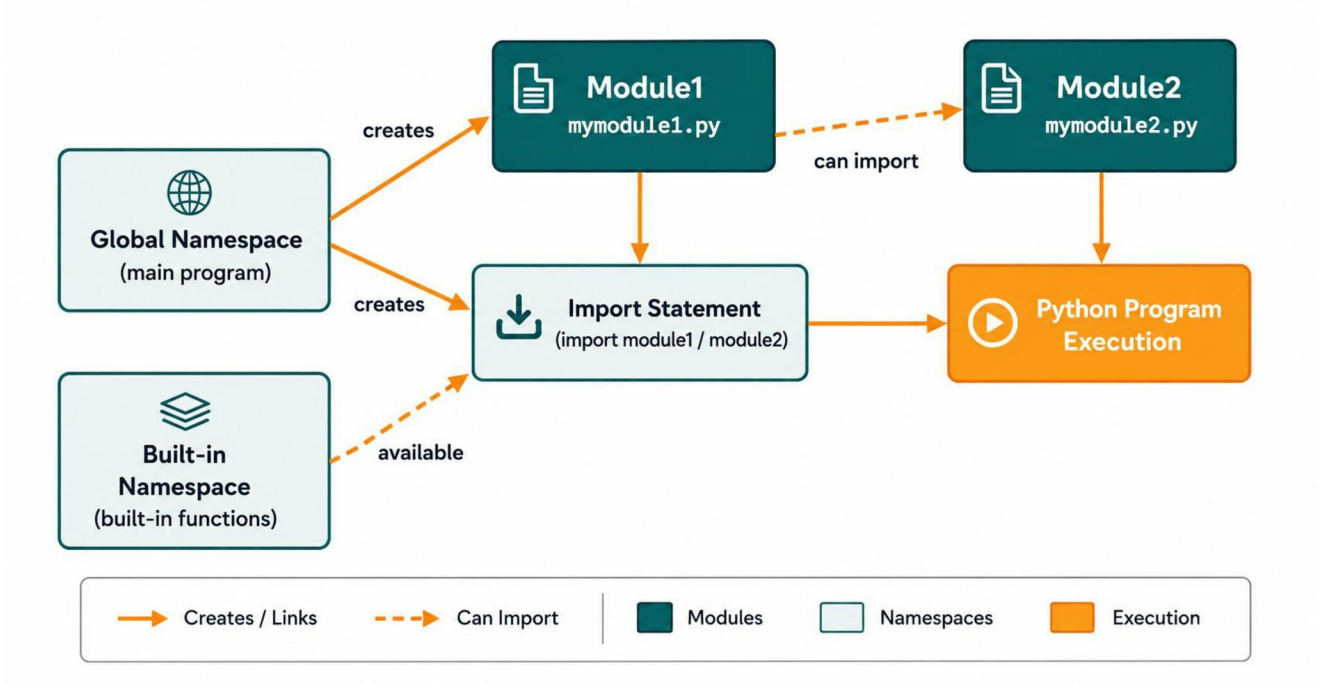
A Few More Handy Functions

Function	Purpose	Example
`np.sort()`	Returns a sorted copy	np.sort(doses)
`np.where()`	Finds positions matching a condition	np.where(doses > 600)
`reshape()`	Changes array shape (same data)	arr.reshape(2,3)
`np.round()`	Rounds all elements	np.round(arr, 2)

```
bp_readings = np.array([118, 145, 132,
128, 160, 110])
high_bp_positions =
np.where(bp_readings > 130)
print(high_bp_positions)      #
(array([1, 2, 4]),)
print(bp_readings[high_bp_positions])
# [145 132 160]
```

Reading and Writing CSV Files

CSV (Comma-Separated Values) is a plain-text format for storing tabular data - each line is a row, and commas separate the columns. Very commonly used for exporting patient/drug records from spreadsheets or hospital systems.



Using the built-in csv module

```
import csv

# Reading a CSV file
with open('patients.csv', 'r') as file:
    reader = csv.reader(file)
    for row in reader:
        print(row)

# Writing to a CSV file
with open('output.csv', 'w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(['Name', 'Age', 'Dose'])
    writer.writerow(['Ravi', 32, 500])
```

Using pandas (easier for larger data)

```
import pandas as pd
df = pd.read_csv('patients.csv')     # read CSV into a DataFrame
print(df.head())                     # preview first 5 rows
df.to_csv('output.csv', index=False) # write DataFrame back to CSV
```

DictReader / DictWriter (reading rows as dictionaries)

csv.DictReader treats the first row as column headers and returns each row as a dictionary - often more convenient than a plain list.

```
import csv
with open('patients.csv', 'r') as file:
    reader = csv.DictReader(file)
    for row in reader:
        print(row['Name'], row['Dose_mg'])
```

Handling Missing Files Safely

```
try:
    with open('patients.csv', 'r') as file:
        data = file.read()
except FileNotFoundError:
    print('Error: patients.csv not found!')
```

□ Common CSV Pitfalls

Forgetting newline='' when writing on Windows can insert blank lines between rows. Also, all values read from a CSV are TEXT (strings) by default - cast numeric columns with int()/float() before calculating.

Understanding Structured Healthcare Datasets

A structured dataset organises data into rows (records) and columns (fields/attributes), like a spreadsheet or database table. Each row is usually one patient, sample, or observation.

Column (Field)	Example Value	Data Type
`Patient_ID`	P1024	String
`Age`	45	Integer
`Drug_Name`	Metformin	String
`Dose_mg`	500	Integer
`BP_Systolic`	128	Integer
`Adverse_Reaction`	False	Boolean

□ Why Structure Matters

Structured data can be easily filtered, sorted, grouped, and analysed with Pandas/NumPy - e.g., 'find all patients above 60 on Metformin' - which is very hard to do with unorganised text.

Rows, Columns & Dataset Shape

A dataset's shape is described as (number_of_rows, number_of_columns) - e.g. (500, 6) means 500 patient records with 6 fields each. In pandas, df.shape gives this directly.

Handling Missing / Dirty Data

Issue	Example	Pandas Fix
`Missing value`	Age = NaN / blank	df.dropna() or df.fillna(0)
`Wrong data type`	'500' stored as text	df['col'] = df['col'].astype(int)
`Duplicate record`	Same Patient_ID twice	df.drop_duplicates()

Sample Healthcare Dataset

Patient_ID	Age	Drug_Name	Dose_mg	BP_Systolic
`P1001`	34	Metformin	500	122
`P1002`	58	Amlodipine	5	138

Patient_ID	Age	Drug_Name	Dose_mg	BP_Systolic
P1003	45	Metformin	1000	128
P1004	67	Atorvastatin	20	145

Simple Grouping Without Pandas

Even without pandas, a dictionary can group and count values from a small dataset - useful for a quick summary.

```
records = [
    {'drug': 'Metformin', 'age': 34},
    {'drug': 'Amlodipine', 'age': 58},
    {'drug': 'Metformin', 'age': 45},
]
drug_counts = {}
for r in records:
    drug_counts[r['drug']] = drug_counts.get(r['drug'], 0) + 1
print(drug_counts)   # {'Metformin': 2, 'Amlodipine': 1}
```

Importing & Working with Pharma Datasets

Combining CSV handling with lists, dictionaries and basic NumPy/Pandas operations lets you build a small working analysis on a real pharmacy dataset.

```
import pandas as pd

df = pd.read_csv('drug_stock.csv')
print(df.columns)              # list of column names
print(df['Stock_Qty'].sum())   # total stock across all drugs

low_stock = df[df['Stock_Qty'] < 20]     # basic filtering
print(low_stock[['Drug_Name', 'Stock_Qty']])

avg_price = df['Price'].mean()
print('Average price:', round(avg_price, 2))
```

Sample Output:

```
Index(['Drug_Name', 'Stock_Qty', 'Price'], dtype='object')
845
  Drug_Name  Stock_Qty
3  Aspirin          12
Average price: 45.75
```

□ Concepts Used

pd.read_csv() to import data, dictionary-like column access df['col'], basic filtering with a condition, and NumPy-backed aggregate functions (.sum(), .mean()) - a full revision of this unit.

Combining Lists, Dicts & NumPy - Inventory Report

A second worked example that builds a small pharmacy inventory summary from raw data, without needing pandas.

```
inventory = [
    {'name': 'Paracetamol', 'qty': 120, 'price': 2.5},
    {'name': 'Ibuprofen',   'qty': 15,  'price': 3.0},
    {'name': 'Aspirin',     'qty': 8,   'price': 1.8},
]

low_stock_items = [item['name'] for item in inventory if item['qty'] < 20]
total_value = sum(item['qty'] * item['price'] for item in inventory)

print('Low stock:', low_stock_items)
print('Total inventory value: Rs.', round(total_value, 2))
```

Sample Output:

```
Low stock: ['Ibuprofen', 'Aspirin']
Total inventory value: Rs. 344.4
```

Quick Reference Cheat Sheet

Category	Function / Method	Purpose
List	append(), insert(), pop(), sort()	Add / remove / order items
Tuple	count(), index()	Read-only lookups
Dict	get(), keys(), values(), items(), update()	Key-based access & bulk update
String	split(), join(), strip(), upper()/lower()	Cleaning & splitting text
NumPy	np.array(), np.arange(), mean(), sum()	Fast numeric array maths
CSV	csv.reader/writer, DictReader/DictWriter	Read/write tabular text files
Pandas	read_csv(), to_csv(), df['col']	Import/export & analyse tables

Noteskarts
Smart Notes, Bright Future

Stay Connected with Noteskarts

Join our community for daily updates, free notes, and exam alerts

WhatsApp Channel **Telegram Group** **YouTube Channel**

Thank You for Studying with Noteskarts!

www.noteskarts.com

Smart Notes. Bright Future.