



Notes by PharmacoLearn

Basics of Python Programming for Pharmaceutical Sciences

BP101T — B.Pharm Semester I

Rapid Revision Notes

One-night-pass companion: every topic, only the core

As per PCI B.Pharm (NEP 2020) syllabus

Scan to connect with PharmacoLearn



YouTube
Subscribe



Telegram
Join the channel



Website
Read the notes

© 2026 PharmacoLearn. All rights reserved.

No part of these notes may be reproduced, stored, or transmitted without prior written permission.
sites.google.com/view/pharmacolearn-notes

Unit I — Introduction to Python programming

Installing Python and an IDE

Python is a high-level, interpreted language — it runs your code line by line, no compiling needed. Download it from python.org and tick "Add Python to PATH" during installation (Windows); check with **python --version**. An IDE (Integrated Development Environment) bundles a code editor with a debugger, terminal and package tools in one window. The syllabus names three: Jupyter Notebook (runs in the browser, code in small cells — your main tool for Units III–V), PyCharm (full-featured, heavier), and VS Code (light, fast middle ground).

Mnemonic — DABS: an IDE beats a text editor on Debugging, Autocomplete, Built-in terminal, Syntax highlighting. Notepad gives you none of these.

Variables and data types

A variable is a named box that stores a value. Python is dynamically typed — you never declare the type; Python works it out from the value. Four types to memorise: **int** (whole numbers: **500**), **float** (decimals: **2.5**), **str** (text in quotes: "**Paracetamol**"), **bool** (**True/False** — capitals matter). Check any type with **type(dose_mg)**.

Remember: Variable names are case-sensitive, cannot start with a digit, and **bool** values always start with a capital T or F.

Type casting

Type casting converts one data type into another: **int()**, **float()**, **str()**, **bool()**. You need it because **input()** always returns a string — "**60**" * **10** repeats text, it does not multiply.

Example: **weight = float(input("Enter weight (kg): "))** then **dose = weight * 10** works. Note **int(2.9)** gives **2** (it chops the decimal, does not round).

Core operators

- Arithmetic: **+** **-** ***** **//** **%** ****** — **/** always gives a float (**5 / 2 = 2.5**), **//** drops the decimal (**5 // 2 = 2**), **%** gives the remainder, ****** is power (**height ** 2** for BMI).
- Comparison (give back **True/False**): **==** **!=** **>** **<** **>=** **<=**.
- Logical: **and** **or** **not** — e.g. (**age >= 18**) **and** (**weight >= 50**).

Trap alert: `=` assigns a value, `==` compares two values. Mixing them up is the classic 1-mark error.

Input and output

- Output: **print()** shows values. F-strings make neat output: **print(f"{drug} {dose}mg twice daily")**.
- Input: **input()** reads one line of text from the user — always a string, so cast it before any calculation.

String operations and manipulation

Strings support: **s.strip()** (remove outer spaces), **s.upper()**, **s.lower()**, **s.replace("old","new")**, **"a,b".split(",")** (string to list), **"-".join(list)** (list to string), **len(s)**, **+** (join), ***** (repeat: **"Na" * 3** → **"NaNaNa"**), **in** (membership: **"para" in "paracetamol"** → **True**).

Indexing starts at 0 (**"Aspirin"[0]** is **'A'**); slicing takes a range with the stop excluded (**"Paracetamol"[0:4]** → **'Para'**).

Remember: The string methods appear again in Unit III for cleaning messy dataset text — learn them once, use them twice.

Standard libraries vs third-party libraries

- Standard library — ships with Python, just **import** it: **math** (square roots, logs — handy for pH/half-life), **statistics** (mean, median), **datetime**, **csv**, **random**.
- Third-party libraries — written by the community, installed with pip: **pip install numpy**, **pip install pandas matplotlib**, **pip uninstall numpy**, **pip list** (shows what is installed).

Remember: Standard = comes with Python. Third-party = pip installs it. The pharma trio for this paper: NumPy (numbers), Pandas (tables), Matplotlib (graphs).

Very short questions (2 marks)

- 1 Expand IDE and name the three IDEs in the syllabus.
- 2 What does **input()** return, and why must you cast it?
- 3 State the difference between `/` and `//` with one example each.
- 4 Write the pip commands to install and uninstall NumPy.
- 5 Differentiate between `=` and `==`.

Short questions (5 marks)

- 1 Explain the advantages of IDEs over text editors with examples.
- 2 Describe the four Python data types with one example each, and explain type casting.
- 3 Explain standard vs third-party libraries with examples and the pip commands to manage them.

Long questions (10 marks)

- 1 Describe Python variables and data types, type casting, and input/output operations. Illustrate with a small program that reads a patient's weight and prints a calculated dose.
- 2 Explain string operations in Python (strip, upper, lower, replace, split, join, indexing, slicing) with examples, and state where each is useful in pharmaceutical data work.

How to approach these answers

- Very-short Q1: one line defining IDE + the three names (Jupyter, PyCharm, VS Code).
- Very-short Q2: one line — **input()** returns a string — plus one casting example.
- Very-short Q3: two tiny examples: **5 / 2 = 2.5**, **5 // 2 = 2**.
- Very-short Q4: two commands — install and uninstall.
- Very-short Q5: one line each — assignment vs comparison.
- Short Q1: open with the IDE definition, then 4 points in the DABS order, one example per point, close with one line.
- Short Q2: define variable, list the four types with examples, then explain casting with the **input()** example.
- Short Q3: two short paragraphs (standard, third-party), one table of examples, the four pip commands.
- Long Q1: structure — intro (what Python is), main body under 3 subheads (variables & types; casting; input/output + program), close with one line on why this matters for pharma.
- Long Q2: structure — intro, main body under 3 subheads (case/spacing methods; split-join; indexing-slicing), examples throughout, conclusion linking to Unit III data cleaning.

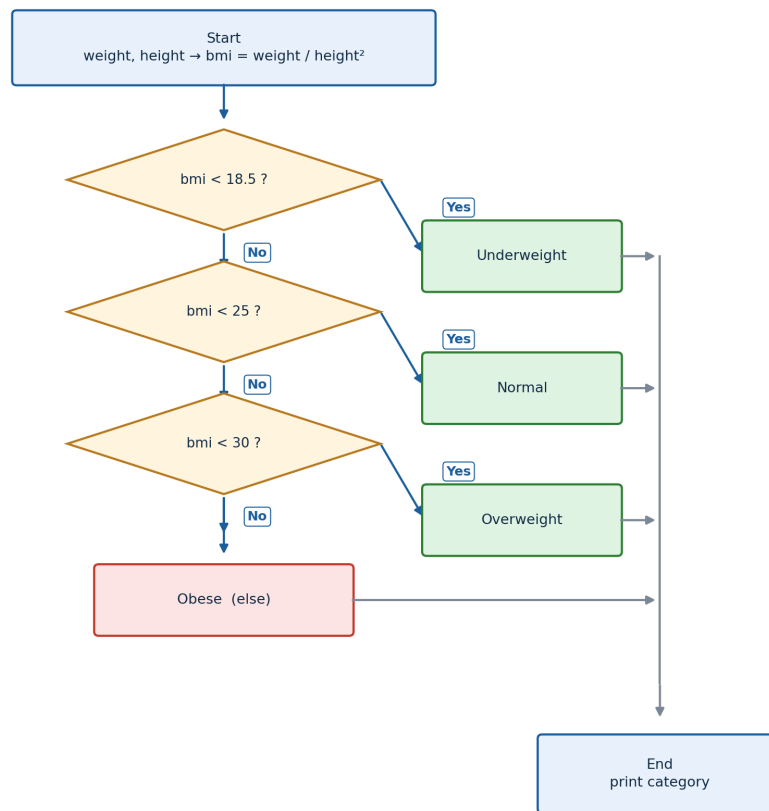
Unit II — Control Structures & Functions

Conditional statements

Conditionals run different code depending on a True/False test. Python uses indentation (spaces, no curly brackets) to mark the block: **if age >= 18:** → adult dose, **else:** → paediatric dose. For more than two cases use **elif**: fever < 37.5 → normal, elif < 39 → mild, **else** → high.

Nested conditions are if-statements inside if-statements — e.g. first check age ≥ 60 , then inside check allergy status. Examiners love patient-eligibility programs with nested if.

if / elif / else decision flow — BMI category program



Once a Yes branch is taken, the later checks are skipped.

if / elif / else decision flow for the BMI category program

Remember: The checks run top to bottom — once one **if/elif** is True, the rest are skipped. Order your conditions from most specific to least.

Loops

Loops repeat a block of code. Use **for** when you know how many rounds: **for i in range(5)** gives 0–4, **for drug in drugs** walks a list. Use **while** when you repeat until something changes: **while count > 0**: — but something inside must move the condition toward False, or you get an infinite loop.

range(n) → 0 to n–1; **range(1, 6)** → 1–5; **range(0, 24, 2)** → even numbers with step 2.

Trap alert: A **while** loop whose condition never becomes False hangs the program. Always check what updates the loop variable.

break and continue

- **break** — exits the loop immediately (stop searching once the drug is found).
- **continue** — skips the current round and jumps to the next one (skip expired batches, process the rest).

Mnemonic: break = break out of the loop entirely; continue = continue with the next round.

Defining and calling functions

A function is a named, reusable block: **def** defines it, calling its name runs it. Arguments are the values you pass in; parameters are the names in the definition; **return** sends a result back (a function without **return** gives **None**). Example: **def calculate_dose(weight_kg, mg_per_kg): return weight_kg * mg_per_kg**, called as **calculate_dose(60, 10)**. You can set defaults: **def calculate_dose(weight_kg, mg_per_kg=10):**.

Modular programs for pharmaceutical applications

"Modular" = split the program into small functions, each doing one job. The two syllabus examples are the most probable coding questions in the paper — learn them by heart.

Dosage (Clark's rule): $\text{child dose} = \text{adult dose} \times (\text{weight} \div 70)$ — **def clarks_dose(adult_dose_mg, weight_kg): return adult_dose_mg * (weight_kg / 70)**.

BMI: **bmi = weight_kg / (height_m ** 2)**, then classify with if-elif-else: < 18.5 underweight, < 25 normal, < 30 overweight, else obese. Format to one decimal with **{bmi:.1f}**.

Remember: BMI cut-offs 18.5 / 25 / 30 are WHO categories — memorise the numbers, not just the code.

Very short questions (2 marks)

- 1 What is the role of indentation in Python conditional statements?
- 2 Differentiate between **break** and **continue** in one line each.
- 3 What does **return** do in a function?
- 4 State Clark's rule as a formula.
- 5 What is an infinite loop, and how do you avoid it?

Short questions (5 marks)

- 1 Explain if, if-else and if-elif-else with a fever-classification example.
- 2 Differentiate between for loop and while loop with examples.
- 3 Explain defining and calling functions: arguments, parameters, return values and default arguments.

Long questions (10 marks)

- 1 Write a modular Python program to calculate BMI and classify the result, explaining each part. Include the flowchart logic for the classification.
- 2 Write a modular Python program for paediatric dose calculation using Clark's rule. Explain why functions make the program modular and reusable.

How to approach these answers

- Very-short Q1: one line — indentation marks the block, no brackets in Python.
- Very-short Q2: break exits the loop; continue skips to the next round.
- Very-short Q3: one line — sends a value back to the caller and ends the function.
- Very-short Q4: the formula only — child dose = adult dose × (weight / 70).
- Very-short Q5: one line definition + one line on updating the loop variable.
- Short Q1: define conditionals, then the fever example showing all three forms, close with the indentation point.
- Short Q2: one paragraph + one code example each, then the "known rounds vs until-condition" rule.
- Short Q3: define function, then arguments/parameters/**return**/defaults with one running example.
- Long Q1: structure — intro (why BMI matters in pharmacy), main body under 3 subheads (the formula; the classification code; the flowchart), conclusion with the 18.5/25/30 cut-offs. Reference the diagram.
- Long Q2: structure — intro (Clark's rule statement), main body under 3 subheads (the **clarks_dose** function; input/output with casting; modularity benefits), conclusion.

Unit III — Data Structures & File Handling

Lists, tuples and dictionaries

Three ways to store collections:

- List — ordered, changeable, square brackets: **drugs = ["Aspirin", "Paracetamol"]**; **drugs.append("Ibuprofen")** adds, **drugs[1] = "New"** edits.
- Tuple — ordered, unchangeable, round brackets: **tablet = ("Aspirin", 75, "mg")** — use for fixed identity data you never want edited by accident.
- Dictionary — key-value pairs, curly braces: **patient = {"id": "P001", "drug": "Aspirin"}** — the natural shape of one patient's record; read with **patient["drug"]**, update with **patient["dose_mg"] = 150**.

Feature	List	Tuple	Dictionary
Brackets	[]	()	{ }
Changeable	Yes	No	Yes (values)
Accessed by	Index	Index	Key
Pharma use	Drug stock list	Fixed (name, strength)	Patient record

Mnemonic — LTD: List changes, Tuple is tied (fixed), Dictionary is by key. Learn the table above — it is a favourite 5-mark question.

Indexing and slicing lists

Indexing starts at 0; negatives count from the end (**drugs[-1]** = last item). Slicing **[start:stop]** includes start, excludes stop: **drugs[0:2]** → first two items, **drugs[1:]** → from index 1 onward.

Handy operations: **len(stock)**, **stock.append(x)**, **stock.remove(x)**, **stock.sort()**, **"Aspirin" in drugs**, and for dictionaries **patient.keys()**, **patient.values()**, **patient.items()**, **patient.get("allergy", "None")** (safe lookup with a default).

String clean-up for data work

The Unit I string methods earn their keep cleaning messy dataset text: **raw.strip().lower()** fixes **"ASPIRIN 75mg"**, **.replace("75mg", "150mg")** corrects values, **.split(",")** breaks one line into fields. Typical exam task: clean a list of drug names (strip spaces, lowercase).

Introduction to NumPy arrays

NumPy is the third-party library for fast numerical work. Its array is like a list where every element is the same type — and maths applies to all elements at once, no loops: **conc = np.array([0, 5, 12])**, then **conc * 2** and **conc.mean()** just work.

Creation: **np.zeros(5)**, **np.ones(4)**, **np.arange(0, 10, 2)** (like **range()**), **np.linspace(0, 1, 5)** (evenly spaced). Know **arr.shape**, **arr.dtype**, **arr.size**.

Remember: NumPy vs lists — arrays are faster, use less memory, and do element-wise maths directly; lists are more flexible (mixed types) but need loops for calculations.

Reading and writing CSV files

CSV (Comma-Separated Values) is the simplest table format — rows are lines, commas separate columns; Excel opens it, and every dataset in Units III–V uses it.

The exam pattern uses the **csv** module: **with open("patients.csv", "w", newline="") as f: + csv.writer(f)** and **writer.writerow([...])** to write; **with open("patients.csv") as f: + csv.DictReader(f)** to read each row as a dictionary keyed by column name. **with open(...)** closes the file automatically; **newline=""** avoids blank lines on Windows.

Structured healthcare datasets

A structured dataset = rows and columns: each row is one record (one patient, one batch, one PK sample), each column is one variable (age, dose, plasma concentration). Before analysing any dataset, ask: what does each column mean, what are the units, are there missing or impossible values (negative age, blank dose)? Putting it together: read a dissolution CSV with **csv.DictReader**, loop rows, and flag batches where **float(r["Dissolution_%"]) < 80** — exactly the shape of the likely exam program.

Remember: Rows = records, columns = variables. Check units and missing values before any calculation.

Very short questions (2 marks)

- 1 State one difference between a list and a tuple with an example.
- 2 What does **drugs[-1]** return, and why?
- 3 What is a CSV file?
- 4 Name two NumPy array-creation functions and what each produces.
- 5 What does **patient.get("allergy", "None")** do?

Short questions (5 marks)

- 1 Differentiate between list, tuple and dictionary with examples.
- 2 Explain indexing and slicing in Python with examples.
- 3 Write a program to read a CSV file of dissolution data and display batches with dissolution below 80%.

Long questions (10 marks)

- 1 Explain NumPy arrays: creation functions, element-wise arithmetic operations and attributes. Compare NumPy arrays with Python lists.
- 2 Explain reading and writing CSV files in Python with the **csv** module, and describe what a structured healthcare dataset is and how you check it before analysis.

How to approach these answers

- Very-short Q1: one line — lists are changeable, tuples are not, with one example.
- Very-short Q2: one line — the last element, because negative indexes count from the end.

- Very-short Q3: one line — comma-separated rows-and-columns text file readable by Excel.
- Very-short Q4: two functions, one line each (e.g. **zeros**, **arange**).
- Very-short Q5: one line — safe key lookup returning "None" if the key is missing.
- Short Q1: the LTD table from the notes + one example per structure.
- Short Q2: define indexing (0-based, negatives), define slicing (**[start:stop]**, stop excluded), one example each.
- Short Q3: write the **with open** + **DictReader** pattern, the loop with the **< 80** condition, and the print line.
- Long Q1: structure — intro (why NumPy exists), main body under 3 subheads (creation; element-wise maths + attributes; NumPy vs lists table), conclusion.
- Long Q2: structure — intro (CSV defined), main body under 3 subheads (writing with **csv.writer**; reading with **DictReader**; structured datasets and the pre-analysis checks), conclusion.

Unit IV — Data Handling with Pandas

Introducing Pandas

Pandas is the library for labelled tables — patient records, PK study results, ADR (Adverse Drug Reaction) reports. NumPy handles raw numbers; Pandas knows column names, handles missing values, and makes filtering and grouping one-liners. Always import as **pd: import pandas as pd**. Excel reading needs **pip install openpyxl**.

Series and DataFrame

Two structures: a Series is one-dimensional — one labelled column (**doses = pd.Series([500, 250], index=["P001", "P002"])**), read by label: **doses["P002"]**); a DataFrame is two-dimensional — the full rows-and-columns table you will use most of the time, built from a dictionary: **df = pd.DataFrame(data)**.

Remember: A DataFrame is a table; each column of it is a Series.

Reading CSV and Excel files

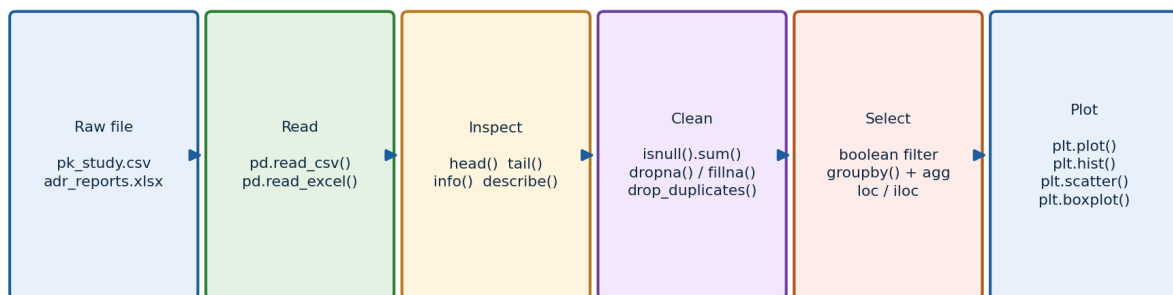
pk = pd.read_csv("pk_study.csv") for PK study data, **adr = pd.read_excel("adr_reports.xlsx")** for ADR reports — each returns a DataFrame. Two practical points: keep the file in the same folder as your notebook (wrong paths are the commonest error), and Excel needs **openpyxl** installed.

Inspecting datasets

Always look at a new dataset first — the four-function toolkit: **df.head()** (first 5 rows), **df.tail()** (last 5), **df.info()** (column names, types, non-missing counts — reveals dirty columns), **df.describe()** (count, mean, std, min, max for numeric columns).

Mnemonic — HTID: "How To Inspect Data" — head, tail, info, describe. A literal likely exam question: "explain four functions used to inspect a dataset."

Pandas data-analysis workflow (Units IV-V)



Golden rule: inspect before you clean, clean before you analyse.

Missing values show as NaN — find with `isnull().sum()`, fix with `dropna()` or `fillna()`.

Pandas data-analysis workflow: read, inspect, clean, select, plot

Cleaning data and handling missing values

Real datasets have gaps — a missed blood sample, an unrecorded age — shown as NaN (Not a Number). Find them with **df.isnull().sum()**; fix with **df.dropna()** (delete rows) or **df.fillna()** (fill gaps — e.g. **df["Age"].fillna(df["Age"].mean())**, the textbook fill for numeric columns). Also **df.drop_duplicates()** and **df["Dose_mg"].astype(int)** to fix wrong types, plus **.str.lower()** / **.str.strip()** for text columns.

Remember: **dropna()** when missing rows are few and unimportant; **fillna()** when you cannot afford to lose rows (dropping 3 of 20 patients hurts).

Filtering and selecting

Filtering picks rows by condition (boolean indexing): **df[df["Age"] >= 18]**; two conditions need brackets each and **&** (AND) / **|** (OR) — not the words **and/or**: **df[(df["Dose_mg"] > 500) & (df["Age"] < 60)]**.

Selecting picks columns or cells: **df["Dose_mg"]** (one column), **df[["Drug", "Dose_mg"]]** (several), **df.loc[0, "Drug"]** (by label), **df.iloc[0:3, 0:2]** (by position).

Trap alert: **loc** = labels, **iloc** = integer positions. Missing brackets around each filter condition is the commonest Pandas error.

Grouping and aggregation

Grouping splits data by a category and summarises each group: **df.groupby("Drug")["Dose_mg"].mean()** (mean dose per drug), **df.groupby("Drug")["ADR_Reported"].sum()** (total ADRs per drug — a likely exam one-liner). Combine several summaries with **.agg({"Dose_mg": "mean", "Age": "max", "PatientID": "count"})**. Aggregation functions to know: **mean()**, **sum()**, **count()**, **max()**, **min()**, **std()**.

Very short questions (2 marks)

- 1 Differentiate between a Pandas Series and a DataFrame in one line each.
- 2 What does **df.info()** tell you?
- 3 How are missing values shown in Pandas, and which function counts them per column?
- 4 Differentiate between **loc** and **iloc**.
- 5 Which library must be installed to read Excel files with Pandas?

Short questions (5 marks)

- 1 Explain any four functions used to inspect a dataset in Pandas.
- 2 Explain how missing values are handled in Pandas (**isnull**, **dropna**, **fillna**).
- 3 A dataset has columns Drug and ADR_Reported. Write Pandas code to find the total ADRs reported for each drug, and explain it.

Long questions (10 marks)

- 1 Describe the Pandas data-analysis workflow (read → inspect → clean → select → plot) with the key functions at each stage. Include the workflow diagram.
- 2 Explain filtering, selecting, grouping and aggregation in Pandas with examples, covering **loc** vs **iloc** and the **&** / **|** operators.

How to approach these answers

- Very-short Q1: Series = one labelled column; DataFrame = full rows-and-columns table.
- Very-short Q2: one line — column names, data types and non-missing counts.
- Very-short Q3: NaN; counted with **df.isnull().sum()**.
- Very-short Q4: **loc** selects by label, **iloc** by integer position.
- Very-short Q5: one word — **openpyxl**.

- Short Q1: the HTID mnemonic, one line of purpose per function.
- Short Q2: define NaN, then find (**isnull().sum()**), remove (**dropna()**), fill (**fillna()** with mean example), and when to choose each.
- Short Q3: the one-liner **df.groupby("Drug")["ADR_Reported"].sum()** plus two lines explaining group-then-sum.
- Long Q1: structure — intro (why the order matters), main body under 3 subheads (read + inspect; clean; select → plot), the workflow diagram, conclusion with the golden rule.
- Long Q2: structure — intro, main body under 3 subheads (filtering with boolean indexing; selecting with loc/iloc; grouping + aggregation), examples throughout, conclusion.

Unit V — Data Visualization with Matplotlib

Introducing Matplotlib

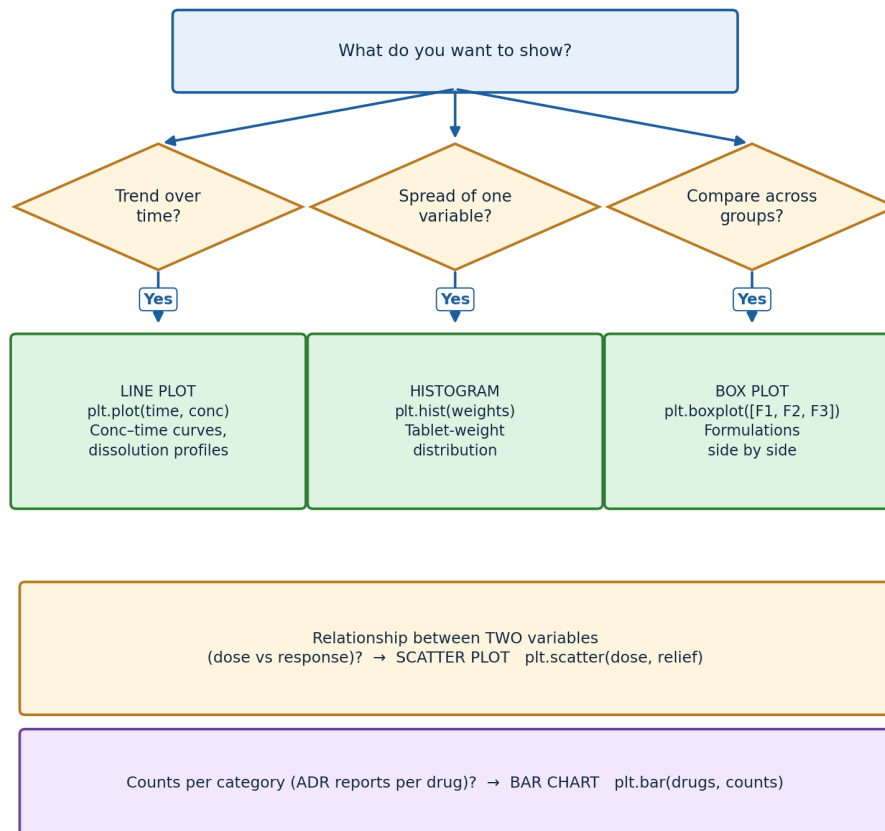
Matplotlib is Python's core plotting library — in pharmacy, the graph is the result: a concentration-time curve is the output of a PK study, a dissolution profile compares formulations. Import as **plt**: **import matplotlib.pyplot as plt**. The pattern is always plot → label → show: **plt.plot(x, y)** then **plt.show()** (in Jupyter the plot appears below the cell, so **show()** is optional).

The four plot types

- Line plot **plt.plot(time, conc)** — trend over a continuous variable (usually time); the workhorse of pharma graphs.
- Histogram **plt.hist(weights, bins=5)** — distribution of one variable (tablet weights); **bins** sets the number of bars.
- Scatter plot **plt.scatter(dose, relief)** — relationship between two variables; an upward trend means positive dose-response.
- Box plot **plt.boxplot([f1, f2, f3], labels=["F1", "F2", "F3"])** — compares distributions across groups in one view.

Mnemonic — LHSB: Line = trend, Histogram = spread, Scatter = two-variable link, Box = groups compared.

Which Matplotlib plot should I use?



Which Matplotlib plot should I use? A decision tree for the four plot types

Labelling axes, titles and legends

An unlabeled graph earns no marks — exam or real report. The checklist: **`plt.xlabel("Time (hours)")`** and **`plt.ylabel("Plasma conc. (µg/mL)")`** with units, **`plt.title(...)`**, **`plt.legend()`** for multiple series (each line named with **`label=`** in the plot call), and **`plt.grid(True)`** to make values readable.

Remember: Every plot needs x-label + units, y-label + units, title, legend (if more than one series). Miss the units and you lose the mark.

Visualizing pharmaceutical datasets

- Concentration-time curves (oral vs IV) — two line plots: the oral curve rises to a peak then falls; the IV bolus curve starts highest at time zero and only falls.
- ADR reporting rates across drugs — a bar chart (one bar per drug) is the natural choice for counts across categories: **`plt.bar(drugs, adr_counts)`**.
- Dissolution profiles — % dissolved vs time, one line per formulation — exactly how you visually compare a test formulation against a reference.

Remember: The syllabus names line, histogram, scatter and box plots. Bar charts are the standard choice for category counts like ADR rates — know them anyway.

Scientific interpretation of plots

Examiners want you to read the graph, not just code it:

- Conc-time curve: the oral peak height is C_{max} , its time is T_{max} ; the IV curve has no absorption phase, so it starts at maximum and declines; area under the curve $\approx$ total drug exposure (AUC).
- Histogram (tablet weights): narrow and symmetric around target = good uniformity; wide or skewed = manufacturing problem.
- Scatter (dose vs response): upward trend = dose-response relationship; random cloud = no clear link.
- Box plot (formulations): compare medians (which dissolves most?), box heights (which is most consistent?), dots outside whiskers = possible outliers.

Exam tip: "Plot the given conc-time data for oral and IV and interpret the curves" (5 marks) = code + two or three interpretation lines like the ones above.

Very short questions (2 marks)

- 1 Which Matplotlib function draws a line plot, and what is it used for?
- 2 State the difference between a histogram and a bar chart.
- 3 What do C_{max} and T_{max} mean on a concentration-time curve?
- 4 In a box plot, what do the line inside the box and dots outside the whiskers show?
- 5 List the four labelling steps every Matplotlib plot needs.

Short questions (5 marks)

- 1 Describe the four Matplotlib plot types and when each is used, with one pharma example per type.
- 2 Explain how to label a Matplotlib plot (axes, title, legend, grid) with the relevant functions.
- 3 Explain how to read a concentration-time curve for oral vs IV administration (C_{max} , T_{max} , absorption phase).

Long questions (10 marks)

- 1 Plot concentration-time data for oral and IV administration using Matplotlib and interpret the curves scientifically. Include the plot-chooser logic for why a line plot is used.

- 2 Explain how histograms, scatter plots and box plots are used and interpreted in pharmaceutical analysis, with one example each (tablet weights, dose-response, formulation comparison).

How to approach these answers

- Very-short Q1: **plt.plot()** — one line on trends over time.
- Very-short Q2: histogram = distribution of one variable; bar chart = values compared across categories.
- Very-short Q3: C_{max} = peak concentration, T_{max} = time of the peak.
- Very-short Q4: line = median; dots = possible outliers.
- Very-short Q5: x-label + units, y-label + units, title, legend — one line listing all four.
- Short Q1: the LHSB mnemonic, one paragraph + one pharma example per plot.
- Short Q2: go through xlabel/ylabel/title/legend/grid with one code snippet each.
- Short Q3: describe the two curves' shapes, define C_{max}/T_{max}, explain the missing absorption phase in IV.
- Long Q1: structure — intro (why conc-time curves matter), main body under 3 subheads (the plotting code; the two curves' shapes; scientific interpretation), conclusion. Reference the plot-chooser diagram.
- Long Q2: structure — intro, main body under 3 subheads (histogram + tablet weights; scatter + dose-response; box plot + formulations), interpretation lines for each, conclusion.