
✅ **How You Should Learn Python (Step-by-Step Roadmap)**

Stage 1 — Foundations (2–3 weeks)

Focus only on **core fundamentals**. Do not jump to frameworks or AI yet.

Learn:

- What is Python, installation (Python + VS Code)
- Variables & Data Types
- Input / Output
- Operators
- Conditional Statements (if / else)
- Loops (for, while)
- Functions
- Basic Error Handling

Stage 2 — Data Handling (2–3 weeks)

Now move to real-world data handling.

Learn:

- Lists, Tuples, Sets, Dictionaries
- Strings (very important)
- File Handling (read/write files)
- Modules & Libraries
- List Comprehension
- Lambda & Map / Filter / Reduce (basic understanding)

PRACTISE

Below are 10 clear, beginner-friendly Python conditional statement practice questions. Each question follows a similar, consistent format and gradually increases in complexity while staying suitable for learners.

1. Write a Python program that checks whether a given number is positive, negative, or zero and prints an appropriate message.
2. Write a Python program that checks whether a given number is even or odd and displays the result.
3. Write a Python program that takes a user's age and determines whether they are a child, teenager, adult, or senior citizen.
4. Write a Python program that checks whether a given character is an uppercase letter, lowercase letter, digit, or special character.
5. Write a Python program that checks whether a given number is divisible by 3 and 5, only 3, only 5, or not divisible by either.
6. Write a Python program that compares two numbers and prints whether the first number is greater than, less than, or equal to the second number.
7. Write a Python program that checks whether a given password length is "Weak", "Medium", or "Strong" based on these rules:
    - Less than 6 characters → Weak
    - 6 to 10 characters → Medium
    - More than 10 characters → Strong
8. Write a Python program that checks whether a user-entered temperature indicates "Cold", "Warm", or "Hot" weather based on suitable conditions.
9. Write a Python program that asks for a student's marks and prints the grade category using if-elif-else (for example: A, B, C, D, or Fail).
10. Write a Python program that checks whether a number is within a specified range (for example, between 1 and 100) and prints whether it lies inside, outside, or exactly on the boundary of the range.

Below are 10 clear, beginner-friendly Python loop practice questions. They follow a similar structure, gradually increase in difficulty, and require practical coding solutions using both for and while loops.

1. Write a Python program using a **for loop** to print all numbers from 1 to 50.
2. Write a Python program using a **for loop** to print the multiplication table of a number entered by the user.
3. Write a Python program using a **while loop** to print numbers from 1 to 10.
4. Write a Python program using a **for loop** to calculate and print the sum of the first 10 natural numbers.
5. Write a Python program using a **while loop** to repeatedly ask the user for a number and stop when the user enters 0.
6. Write a Python program using a **for loop** to print each character of a string on a new line.
7. Write a Python program using a **for loop** to find and print all odd numbers between 1 and 100.
8. Write a Python program using a **while loop** to reverse a given number and display the reversed result.
9. Write a Python program using a **for loop** to count how many vowels are in a given string.
10. Write a Python program using a **nested loop** to print a right-angled triangle pattern made of stars (*) with 5 rows.

Below are 10 clear, beginner-to-intermediate practice questions focused on **Python functions for data engineering contexts**. They are similar in structure, progressively build skills, and emphasize defining functions, arguments, return values, and lambda usage.

1. Write a Python function that takes a list of numbers as input and returns the average value of the list.
2. Write a Python function that accepts a list of integers and returns a new list that contains only the even numbers.
3. Write a Python function that takes a list of strings representing dataset column names and returns the list in uppercase form.
4. Write a Python function that accepts a list of numbers and returns the maximum and minimum values as a tuple.
5. Write a Python function that takes a list of dictionaries representing employee records and returns only those records where the salary is greater than a given value passed as an argument.
6. Write a Python function that accepts a list of numbers and returns a new list in which each value is squared. Then rewrite the same solution using a lambda function with map().
7. Write a Python function that accepts a CSV filename and returns the total number of rows in the file. (Assume the learner uses Python file handling.)
8. Write a Python function that accepts a list of transaction amounts and returns the sum of only positive values, ignoring negative ones.
9. Write a Python function that takes a dictionary of column names and their data types and prints each column name with its corresponding type in a formatted way, then returns the count of columns.
10. Write a Python function that accepts a list of numbers and returns a new sorted list using a lambda function as the sorting key. Explain through code comments how the lambda is used in sorting.

Below are 10 well-structured, beginner-to-intermediate practice questions focused on **Python error handling for data engineering scenarios**. They are similar in style, build progressively, and emphasize practical application.

1. Write a Python program that attempts to open a dataset file using a try-except block and prints a user-friendly message if the file is not found.
2. Write a Python program that reads numbers from a list and handles the ValueError that occurs if a non-numeric value is encountered, printing a suitable error message.
3. Write a Python program that divides two numbers entered by the user and handles a ZeroDivisionError, printing a meaningful error explanation.
4. Write a Python program that tries to access a column from a dataset dictionary and handles the KeyError if the column does not exist.
5. Write a Python program that uses a try-except-else block to load a file: print a success message if the file loads correctly, otherwise handle the exception gracefully.
6. Write a Python program that uses a try-except-finally block to connect to a data source, handle any connection failure, and ensure that the connection is always closed in the finally block.
7. Write a Python program that raises a ValueError if a dataset contains negative values where only positive numbers are allowed, and handles that exception properly.
8. Write a Python program that defines a **custom exception** called InvalidFileFormatError and raises it when a file does not have the required .csv extension.
9. Write a Python function that processes a list of records and uses try-except to handle any unexpected data type errors, ensuring that processing continues for remaining records.
10. Write a Python program that chains multiple exception types (FileNotFoundError, ValueError, and TypeError) in a single try-except block and prints different custom messages for each, simulating a real data pipeline error scenario.

Below are 10 progressively challenging practice questions on Python Lists tailored for data engineering use cases. They emphasize data manipulation, storage efficiency, and analytical thinking.

1. You have a dataset containing daily website visitors: [120, 340, 560, 230, 450]. How would you access the first and last elements from this list?
2. Create a Python list named temperatures to store 7 daily temperature readings collected from a sensor.
3. Given a list of integers representing sales data, [23, 45, 12, 67, 34, 89], how would you find the highest and lowest sales values using built-in functions?
4. Suppose you have a list of user IDs: [101, 102, 103, 104, 105]. How would you add a new user ID 106 to the end of the list and insert 100 at the beginning?
5. You receive raw streaming sensor values as [10, 20, 30, 40, 50]. How would you update the 3rd element to 35 and remove the last element from the list?
6. Given a dataset list [12, 15, 7, 18, 10, 22, 5], how would you extract only the first five values using list slicing?
7. You have a list of API response times in milliseconds: [120, 340, 90, 450, 200, 600]. How would you filter and create a new list containing only response times greater than 200 ms?
8. A company stores daily revenue values in a list. Write a Python expression to calculate the total revenue using sum() and the average revenue using list length.
9. You receive a CSV column as list values: ["India", "USA", "India", "UK", "USA", "India"]. How would you count how many times "India" appears in the list?
10. You have two lists of data pipeline execution times:
    pipeline1 = [30, 40, 35, 50]
    pipeline2 = [25, 45, 32, 48]
    How would you merge them into a single list and then sort the combined execution times in ascending order?

Below are 10 progressively structured practice questions on Python **tuples**, designed specifically for data engineering learning contexts. They focus on tuple creation, immutability, indexing, unpacking, and real-world data handling use cases.

1. How do you create a tuple in Python? Write an example of a tuple that stores daily temperature readings for three cities.
2. What are the key differences between tuples and lists in Python, and why might tuples be preferred in certain data engineering scenarios?
3. Given a tuple data = (101, "CustomerA", 45000), how would you access the first and last elements? Explain using indexing.
4. You receive immutable configuration settings for a data pipeline in the form of a tuple: config = ("AWS", "ap-south-1", "t2.medium"). Explain why immutability is beneficial in this scenario.
5. Given a tuple metrics = (120, 340, 560, 230, 450), how would you extract only the first three values using slicing?
6. Explain tuple unpacking with a data engineering example where a tuple contains (table_name, row_count, last_updated) and you need to assign them to individual variables.
7. Given two tuples representing batch job execution times:
   batch1 = (30, 45, 60) and batch2 = (25, 50, 55)
   How would you combine them into a single tuple?
8. A tuple stores read-only credentials:
   credentials = ("admin", "db_user", "readonly_access")
   Demonstrate what happens if you try to modify an element. Why is this behavior important in secure systems?
9. You have a list of tuples representing user records:
   users = [(101, "Amit"), (102, "Riya"), (103, "John")]
   How would you access only the usernames from this structure?
10. In a data pipeline, you receive a tuple containing sensor data in the format (sensor_id, location, value). Provide a real-world example of how tuples help maintain structured, ordered, and immutable dataset records.

Below are 10 progressively structured practice questions on Python **sets**, designed specifically for data engineering learning contexts. They focus on applying set theory concepts to real-world data workflows such as deduplication, filtering, uniqueness checks, and efficient lookups.

1. You receive a dataset containing user IDs with duplicates:
   user_ids = [101, 102, 103, 101, 104, 102].
   How can you use a set to remove duplicate user IDs?

2. Given two sets representing users of two different applications:
   appA_users = {101, 102, 103, 104}
   appB_users = {103, 104, 105, 106}
   How would you find users who use both applications using set operations?

3. In a data pipeline, you have two datasets:
   clean_data_ids = {1, 2, 3, 4, 5}
   corrupted_data_ids = {4, 5, 6}
   How would you identify records that are clean and not corrupted using a set operation?

4. A data engineer wants to check if all records in a validation dataset exist in a master dataset.
   If validation = {10, 20, 30} and master = {10, 20, 30, 40, 50},
   how would you verify whether validation is a subset of master?

5. You are processing streaming logs where the same IP may appear multiple times.
   Explain how sets can help in counting unique IP addresses efficiently.

6. Given two datasets representing different days of website visitors:
   day1 = { "A", "B", "C", "D" }
   day2 = { "C", "D", "E", "F" }
   How would you determine users who visited on at least one of the days?

7. You have a list of country names where some values repeat.
   Describe how you would convert the list into a set and then back into a list to maintain only unique countries.

8. A pipeline needs to quickly verify whether a specific transaction ID exists in a dataset of 1 million IDs.
   Why is a set more efficient than a list for membership checking in this scenario?

9. You have two sets of sensor IDs:
   active_sensors = {11, 12, 13, 14}

failed_sensors = {13, 15}
How would you identify sensors that are active but not failed?

10. You are given a dataset containing product categories from different sources:
    source1 = {"Electronics", "Clothing", "Home"}
    source2 = {"Home", "Sports", "Toys"}
    Demonstrate how union, intersection, and difference operations could be applied to support decisions in data integration.

Below are **10 progressively structured practice questions on Python Dictionaries**, designed specifically with **data engineering scenarios** in mind. They focus on dictionary creation, key-value access, updates, aggregations, and real-world usage.

1. Create a dictionary named employee_salaries that stores employee names as keys and their salaries as values. How would you retrieve the salary of a specific employee using their name?
2. Suppose you have a dictionary storing product prices:
   products = {"Laptop": 75000, "Phone": 35000, "Tablet": 25000}.
   How would you update the price of "Phone" and add a new product "Smartwatch"?
3. You receive API response data as a dictionary:
   user = {"id": 101, "name": "Amit", "country": "India"}.
   How would you access only the keys and only the values from this dictionary?
4. Given a dictionary of product quantities in a warehouse:
   stock = {"Apples": 50, "Oranges": 30, "Bananas": 20},
   write logic to increase "Apples" stock by 10 and decrease "Bananas" by 5.
5. A data engineer receives a dictionary of dataset statistics like:
   stats = {"rows": 150000, "columns": 28, "null_values": 4500}.
   How would you check if a key like "null_values" exists before accessing it?
6. You have a dictionary of city temperatures:
   temps = {"Mumbai": 32, "Delhi": 28, "Pune": 26, "Chennai": 34}.
   Write a way to find the city with the highest temperature using dictionary operations.
7. Consider the dictionary of pipeline execution times:
   pipelines = {"job1": 35, "job2": 50, "job3": 28}.
   How would you calculate the total and average execution time?
8. You have a list of tuples representing user data:
   [("A101", "Riya"), ("A102", "John"), ("A103", "Amit")].
   How would you convert this list into a dictionary where the ID is the key and the name is the value?
9. In a data pipeline, you receive nested dictionary data such as:
   customer = {"id": 201, "details": {"name": "Rahul", "age": 29, "country": "India"}}.
   How would you access the customer's country value?
10. You have two dictionaries representing monthly revenue from different regions:
    region1 = {"Jan": 12000, "Feb": 15000}

region2 = {"Feb": 10000, "Mar": 18000}
How would you merge them so that common months have their values summed?

Below are **10 concise practice questions on Python strings**, structured for clarity and relevance to **data engineering workflows** such as cleaning data, parsing text, and formatting outputs.

1. Given a log entry string, how would you extract only the first 10 characters using string slicing?
2. You receive column names as "Customer_Name". How would you convert this string to lowercase and replace the underscore with a space?
3. Given a filename "data_pipeline_2025.csv", how would you check if the string ends with .csv?
4. A dataset field contains extra spaces: " Mumbai ". How would you remove leading and trailing spaces using a string method?
5. You receive full names as "Amit Sharma". How would you split this string into first name and last name?
6. A data engineer wants to count how many times "ERROR" appears in a log string. Which string method would you use and how?
7. Given two strings "Data" and "Engineering", how would you concatenate them with a space in between?
8. You get a string input "mumbai" and need it in title case like "Mumbai". Which method would you apply?
9. Suppose you need to mask part of a sensitive string such as "AccountNumber: 9876543210" to show only the last four digits. How would you use slicing to achieve this?
10. You have a string "2025-12-30" representing a date. How would you split this string into year, month, and day components?

Below are **10 clear, practical, and data-engineering–focused practice questions on Python File Handling**, designed to build understanding of reading, writing, appending, working with CSV/JSON, file modes, and safe handling practices:

1. Write a Python function that reads a text file containing log entries and prints only the first 10 lines. Which file mode will you use and why?
2. Create a script that opens a CSV file, reads all rows, and counts how many rows contain the word "ERROR". Explain how you would safely handle file operations.
3. Write a Python program that writes a list of dataset column names into a text file, ensuring that every column name is written on a new line. Which file mode is appropriate?
4. Design a function that appends a new record to an existing CSV file without deleting previous data. Explain why append mode is used here.
5. You are given a JSON file containing database configuration details. Write a program to read the JSON file and print the value of a specific key such as "database_name".
6. Create a script that reads data from one file and writes only unique lines into another file. Explain how this can help in data deduplication tasks.
7. Write a Python program that attempts to open a file containing pipeline execution logs. If the file does not exist, handle the exception gracefully and display an appropriate message.
8. Develop a script that reads a large CSV file line-by-line instead of loading the entire file into memory at once. Explain why this approach is efficient for big data processing.
9. Write a function that writes structured data into a JSON file, ensuring that existing content is not lost unless intentionally overwritten. Which mode would you use?
10. Create a script that opens a dataset file, processes its content, and ensures the file is automatically closed after execution. Explain how the with statement helps in safe file handling.

Below are **10 clear, practical, and data-engineering–focused practice questions on Python Modules & Libraries**. They progressively reinforce understanding of importing, using built-ins, managing external libraries, and leveraging modules in real data workflows.

1. Explain the purpose of the import statement in Python. Provide an example of importing a specific function from a module instead of importing the entire module.
2. What is the difference between a module and a package in Python? Provide one example of each commonly used in data engineering.
3. How would you install third-party libraries such as **Pandas** or **NumPy** using pip, and why are these libraries important for data engineering tasks?
4. Write a Python example demonstrating how to read a CSV file using the **Pandas** library and print the first 5 rows of the dataset.
5. Explain how the os and sys modules can assist a data engineer in handling file paths and environment configurations. Provide a simple example.
6. How can you use the datetime module to add timestamps to data logs in a data pipeline? Write a small example.
7. Describe how to create a **custom Python module** that contains a function for calculating the average of a list of numbers. Explain how to import and use that module in another script.
8. What is the purpose of the math and statistics libraries in Python? Provide one data engineering scenario where each could be applied.
9. In a data engineering workflow, how would you use **JSON** and **CSV** libraries to handle structured data? Provide an example of reading a JSON file.
10. Explain what virtual environments are and why they are important when working with multiple Python libraries in data engineering projects. Provide steps to create and activate one.

Below are **10 clear, practical, and data-engineering–focused practice questions on List Comprehension in Python**. They progressively build understanding of syntax, filtering, transformation, and real-world applications.

1. Given a list of integers representing daily user logins, write a list comprehension to create a new list containing only values greater than 100.
2. You receive a list of strings representing city names with extra spaces (e.g., " Mumbai "). Write a list comprehension to strip whitespace from all city names.
3. A dataset contains temperature readings in Celsius. Using list comprehension, convert all values to Fahrenheit and store them in a new list.
4. Given a list of filenames, use list comprehension to extract only those filenames that end with .csv, representing valid dataset files.
5. You have a list of numbers representing transaction amounts. Write a list comprehension to replace negative values with 0 while keeping positive values unchanged.
6. From a list of API response times in milliseconds, create a new list that contains "High" for values above 300 ms and "Normal" for others using conditional expressions inside a list comprehension.
7. Given a list of email addresses, use list comprehension to extract only the domain part (text after "@") from each email.
8. You receive a list of strings containing numeric values. Using list comprehension, convert all string numbers to integers.
9. From a nested list containing daily sales data for multiple stores, use a nested list comprehension to flatten it into a single consolidated list.
10. You have a list of dictionaries where each dictionary contains user data with a key "status". Using list comprehension, extract only those user records whose status is "active".

Below are five clear, practical, and data-engineering–focused practice questions on **lambda functions in Python**. They emphasize filtering, transformation, sorting, and mapping in real-world scenarios.

1. You have a list of transaction amounts: [1200, -300, 4500, -150, 2000]. Write a lambda-based expression to filter and return only positive transaction values.
2. Given a list of tuples representing (user_id, login_time) such as [(101, 35), (102, 20), (103, 50)], use a lambda function to sort the list based on login_time in ascending order.
3. You receive a list of numbers representing data packet sizes in bytes. Write a lambda function using map() to convert each packet size into kilobytes.
4. A dataset contains a list of usernames. Write a lambda-based filter to return only those usernames that start with "data_", representing data engineering accounts.
5. You have a list of dictionaries representing customers:
   customers = [{"name": "Amit", "age": 28}, {"name": "Riya", "age": 22}, {"name": "John", "age": 35}].
   Use a lambda function to extract only those customers whose age is greater than 25.

Here are **5 clear, data-engineering–focused practice questions on the** map() **function in Python**, similar in structure and relevance to real-world workflows:

1. You have a list of temperatures in Celsius: [25, 30, 35, 40]. How would you use map() to convert them to Fahrenheit and return a list of converted values?
2. A list of strings contains city names such as ["mumbai", "delhi", "pune"]. Write a Python expression using map() to convert each city name to uppercase.
3. You receive a list of numeric strings representing transaction amounts: ["100", "250", "400", "750"]. How would you use map() to convert all the values into integers?
4. Given a list of data processing times in seconds: [2.5, 3.8, 1.2, 4.0], use map() to round each value to the nearest integer.
5. You have a list of dictionaries representing users:
   users = [{"name": "Amit"}, {"name": "Riya"}, {"name": "John"}].
   Use map() to extract only the "name" values into a separate list.

Below are **five clear, practical, and data-engineering–focused Python practice questions on filtering**, designed to be similar in structure, progressively thoughtful, and grounded in real-world workflows.

1. You are given a list of transaction amounts:
   [1200, 450, 50, 9800, 200, 15000].
   Write code to filter and retain only high-value transactions greater than 1,000 using Python filtering techniques.

2. You have a list of dictionaries representing user data:
   [{"name": "Amit", "age": 28}, {"name": "Riya", "age": 17}, {"name": "John", "age": 35}].
   Write Python code to filter and keep only users who are 18 years or older.

3. A log file is read into a list of strings, where each string represents a log entry.
   Filter the logs to extract only entries containing the keyword "ERROR".

4. You have a Pandas DataFrame containing employee records with columns ["name", "department", "salary"].
   Write a filter to select only employees working in the "Data Engineering" department with a salary greater than 80,000.

5. You receive a dataset of email addresses stored in a list.
   Using regex filtering, write Python code to retain only valid emails ending with "@company.com".

Below are **five clear, practical, and data-engineering–focused practice questions on the Python** reduce() **function**, designed with similar structure and intent to reinforce understanding.

1. You are given a list of transaction amounts: [1200, 450, 800, 3000, 150]. Using the reduce() function, write logic to calculate the total transaction value. Explain why reduce() is appropriate here.
2. Consider a list of data processing durations in seconds: [2.5, 3.1, 4.0, 1.8]. Using reduce(), write Python code to find the maximum processing time without using the built-in max() function.
3. You receive a dataset containing batch job execution counts per day: [5, 7, 6, 9, 4]. Using reduce(), write logic to compute the cumulative product of these values and explain a use case where such computation might be useful.
4. You have a list of strings representing log message parts: ["Data", "Processing", "Completed"]. Using reduce(), write a Python expression to concatenate them into a single message with spaces in between.
5. Given a list of dictionaries representing daily revenue:
   [{"revenue": 1200}, {"revenue": 800}, {"revenue": 1500}, {"revenue": 600}],
   use reduce() to compute the total revenue extracted from all dictionaries.