Python – 100+ Lab Exercises (Basic, Intermediate, Advanced)

☐ Basic Level (30+ Exercises)

Objective: Understand core Python syntax, data types, and control flow.

☐ Python Fundamentals

- Print "Hello, World!"
- Accept user input and print it.
- Use `type()` to display variable types.
- Perform basic arithmetic operations.

☐ Variables & Data Types

- Define `int`, `float`, `str`, `bool`, `complex`.
- Use `typecasting` between data types.
- Create and manipulate strings with methods (`.upper()`, `.find()`).
- Work with `len()`, `min()`, `max()`.

☐ Control Structures

- Write a simple `if`, `elif`, `else` condition.
- Use `for` and `while` loops.
- Create a multiplication table.
- Check for prime numbers.
- Implement a basic calculator using `if` statements.

☐ Collections

- Create and manipulate a `list`, `tuple`, `set`, `dictionary`.
- Slice lists and access nested collections.
- Add, remove, and update dictionary items.
- Convert between data types (`list` to `set`, etc.).

☐ Functions

- Write functions with default and keyword arguments.
- Use `*args` and `**kwargs`.
- Create a recursive function (e.g., factorial).
- Use `lambda`, `map()`, `filter()`, and `reduce()`.

☐ Intermediate Level (40+ Exercises)

Objective: Strengthen algorithmic thinking, file handling, OOP, and error handling.

☐ File I/O

- Read from and write to a `.txt` file.
- Count words/lines in a file.
- Parse a CSV file manually.
- Log errors in a file.

☐ String & List Algorithms

- Reverse a string.
- Check for palindrome and anagram.
- Remove duplicates from a list.
- Flatten a nested list.
- Find the most frequent element in a list.

☐ Object-Oriented Programming

- Create a class with attributes and methods.
- Use constructors (`__init__`).
- Inherit from a base class.
- Use method overriding and `super()`.
- Implement encapsulation with private/protected members.

☐ Exception Handling

- Handle `ZeroDivisionError`, `ValueError`, `FileNotFoundError`.
- Use `try-except-finally` blocks.
- Raise custom exceptions.

☐ Modules & Packages

- Import and use built-in modules (`math`, `datetime`, `random`).
- Create a custom module and import it.
- Organize code using packages (`__init__.py`).
- Use `dir()` and `help()`.

☐ Date & Time

- Get the current date and time.
- Calculate difference between two dates.
- Convert timestamp to datetime.

☐ Advanced Level (40+ Exercises)

Objective: Explore web APIs, testing, multithreading, decorators, and project-level development.

☐ Web Scraping & APIs

- Make API calls using `requests`.
- Parse JSON data from API response.
- Web scrape a site using `BeautifulSoup`.
- Automate Google search queries.

☐ Threading & Concurrency

- Create multithreaded programs using `threading`.
- Use `time.sleep()` and thread synchronization.
- Use `asyncio` for async programming.

☐ Decorators & Generators

- Create a function decorator.
- Chain multiple decorators.
- Write a generator using `yield`.
- Create an infinite generator.

☐ Unit Testing

- Write test cases using `unittest`.
- Assert exceptions and edge cases.

- Mock function calls.

☐ Advanced Data Structures

- Implement stack, queue using lists.
- Create a binary search tree (BST).
- Use `collections.deque`, `defaultdict`, `Counter`.
- Create a custom linked list class.

☐ Virtual Environments & PIP

- Create a virtual environment with `venv`.
- Install and freeze dependencies with `pip`.
- Share code using `requirements.txt`.

☐ Capstone Projects

- **Student Record System** – CRUD CLI app with file persistence.
- **Weather Dashboard** – API-based GUI using `tkinter`.
- **Expense Tracker** – CSV-based income/expense manager.
- **Todo App (OOP + File I/O)** – Full-featured local task manager.
- **Blog API** – Flask + SQLite backend service.

☐ Bonus Tracks (Optional)

- Use `Tkinter` to build GUIs.
- Build a Flask or Django app.
- Connect Python to PostgreSQL with `psycopg2`.
- Use `matplotlib` and `pandas` for data analysis.