Lesson Plan – Python Lesson 3 (1 Hour)

Objective

By the end of this lesson, students will:

- Understand tuples, lists, dictionaries, and their differences.
- Write simple classes and use OOP concepts (attributes, methods, inheritance).
- Understand modules, packages, and namespaces.
- Work with files and CSVs using `pathlib` and `csv`.
- Build a modular, persistent **Task Tracker app**.

1. Data Structures (15 min)

Tuples

- Defined with `()`. Immutable (cannot be changed).
- Single-value tuple requires a trailing comma.

```
>>> t1 = (1, 2, 3)
>>> t2 = (1,)
>>> print(type(t2))   # tuple
```

- Can use `tuple()` constructor.
- Support indexing, slicing, `len()`.
- Packing/unpacking:

```
>>> a, b, c = (1, 2, 3)
```

Lists

- Defined with `[]`. Mutable (can be changed).

```
>>> mylist = [1, 2, 3]
>>> mylist[0] = 99
```

- Methods: `.append()`, `.insert()`, `.pop()`, `.extend()`, `.sort()`.
- Functions: `sum()`, `min()`, `max()`.
- List comprehension:

```
>>> squares = [x**2 for x in range(5)]
```

Dictionaries

- Key-value pairs inside `{}`.

```
>>> person = {"name": "Alice", "age": 25}
>>> print(person[“name"])
```

- Methods: `.items()`, `.keys()`, `.values()`.
- Add/remove keys with assignment or `del`.
- Nested dictionaries are possible.

2. Object-Oriented Programming (10 min)

- Objects bundle data (attributes) and behavior (methods).
- Define a class:

```
>>> class Animal:
>>>     def __init__(self, name):
>>>         self.name = name
>>>     def speak(self):
>>>         return “..."
```

- Instantiate:

```
>>> dog = Animal("Fido")
>>> print(dog.name)
```

- Show `__str__` for readable output.
- Inheritance:

```
>>> class Dog(Animal):
>>>     def speak(self):
>>>         return “Woof!"
```

- Use `super()` to call parent methods.

3. Modules & Packages (5 min)

- **Modules** = Python files you can import.
- **Packages** = Folders with multiple modules, require `__init__.py`.
- Syntax:

```
>>> import mymodule
>>> from mymodule import func
>>> import mymodule as mm
```

4. Files & CSVs (10 min)

- File paths with `pathlib`.

```
>>> from pathlib import Path
>>> print(Path.cwd())
```

- Reading and writing:

```
>>> with open("file.txt", "w", encoding="utf-8") as f:
>>>     f.write("hello")
```

- CSV module:

```
>>> import csv
>>> with open("tasks.csv", "w", newline="") as f:
>>>     writer = csv.writer(f)
>>>     writer.writerow(["task", "difficulty"])
```

5. Project: Task Tracker (15 min)

Break into **modules** (`task.py`, `file_manager.py`, `main.py`).

- **`task.py`**: Functions to create/display tasks.
- **`file_manager.py`**: Load/save tasks from CSV.
- **`main.py`**: User interface with menu and loop.

Students build features:

- Add task (with `name`, `difficulty`, `due_date`, `completed`).
- Display all tasks.
- Mark tasks completed.
- Store tasks persistently in `tasks.csv`.

Run project, demonstrate persistence across runs.

Wrap-Up (5 min)

- Recap: tuples/lists/dicts, OOP basics, modules/packages, file handling.
- Homework: extend Task Tracker with categories or priority.

Teacher Guide / Aide – Lesson 3

Timing Breakdown

- Data Structures → 15 min
- OOP → 10 min
- Modules/Packages → 5 min
- File Handling/CSV → 10 min
- Project → 15 min
- Wrap-Up → 5 min

Teaching Tips

- **Analogy**:
  - Tuple = "frozen" list (unchangeable).
  - List = shopping list (changeable).
  - Dict = dictionary (look up words/values).
- For OOP, stress **instances vs classes**.
- Show both **mutable vs immutable** by trying to reassign inside tuple vs list.
- Reinforce `with open(...) as f:` pattern → students often forget to close files.
- During project, scaffold in steps:
  - Create task dict.
  - Write to CSV.
  - Load from CSV.
  - Add menu.

Common Pitfalls

- Forgetting the comma in a single-value tuple.

- Mixing up list `[]` vs dict `{}` vs tuple `()`.
- Misunderstanding `self` → it must always be first parameter in class methods.
- Forgetting `import` statements or wrong module paths.
- CSV writer vs reader confusion (`DictWriter` vs `DictReader`).

Stretch for Fast Learners

- Add deadlines as `datetime` objects.
- Add sorting by due date or difficulty.
- Create subpackage `utils` for helper functions.
- Add error logging with `try/except` writing to a log file.