Lesson Plan – Python Lesson 2 (1 Hour)

Objective

By the end of this lesson, students will:

- Understand how to define and use functions with arguments and return values.
- Use loops (`while`, `for`) to repeat tasks.
- Understand scope and the LEGB rule.
- Use conditionals (`if`, `elif`, `else`) with Boolean logic.
- Handle errors with `try/except`.
- Build a modular project with functions, conditionals, loops, and randomness.

1. Functions (10 min)

- Functions are values; they can be stored in variables.
- Arguments go in, return values come out.
- Introduce `None`.

Examples:

```
>>> def squares(x):
>>>     return x * x
>>>
>>> print(squares(4))   # 16
>>>
>>> def greet():
>>>     print("Hello!")   # no return, returns None
```

- Show whitespace errors (Python requires indentation).

Try:

```
>>> def bad():
>>> print("oops")   # error
```

- Introduce docstrings + `help()`:

```
>>> def greet():
>>>     """This function prints a greeting."""
>>>     print("Hello")
```

2. Loops (10 min)

- **While loops**: repeat while condition is true.
- Infinite loop demo (`while True:`) and stopping with Ctrl+C.
- **For loops**: iterate over collections.

Examples:

```
>>> for letter in "string":
>>>     print(letter)
>>>
>>> for i in range(5):
>>>     print(i)
```

- Nested loops:

```
>>> for i in range(2):
>>>     for j in range(2):
>>>         print(i, j)
```

3. Scope & LEGB Rule (5 min)

- Variables live in different “namespaces.”
- Local, Enclosing, Global, Built-in.
- Show shadowing:

```
>>> x = 10
>>> def func():
>>>     x = 5
>>>     print(x)
```

```
>>> func()   # prints 5
>>> print(x) # prints 10
```

4. Conditionals & Boolean Logic (10 min)

- Comparisons: `==` `!=` `>` `<` `>=` `<=`.
- Lexicographic string comparison.
- Keywords: `and`, `or`, `not`.

Examples:

```
>>> if 5 > 3:
>>>     print("true")
```

```
>>> x = "apple"
>>> y = "banana"
>>> print(x < y)  # True, alphabetical
```

- `if`, `elif`, `else`.
- Nested if blocks.
- `continue`, `break`, `else` in loops.

5. Error Handling (5 min)

- Introduce `try/except`.

```
>>> try:
>>>     x = int("abc")
>>> except ValueError:
>>>     print("Invalid number")
```

6. Random Module (2 min)

- Import random, show `randint()`.

```
>>> import random
>>> print(random.randint(1, 10))
```

7. Project: Modular Task Logger (15 min)

Expand Lesson 1's project:

- Break code into **functions**: `get_task()`, `validate_difficulty()`, `calculate_score()`, `print_task_summary()`.
- Add **conditionals** to validate difficulty (1–5).
- Add **loop** to keep entering tasks.
- Add **try/except** for invalid inputs.
- Add **random motivation bonus**.

Wrap-Up (3 min)

- Recap: functions, loops, scope, conditionals, try/except, randomness.
- Homework: Modify the project to:
    - Keep tasks in a list and print all at the end.
    - Add categories to tasks.

Teacher Guide / Aide – Lesson 2

Timing Breakdown

- Functions → 10 min
- Loops → 10 min
- Scope → 5 min
- Conditionals → 10 min
- Error Handling → 5 min
- Random → 2 min
- Project → 15 min
- Wrap-up → 3 min

Teaching Tips

- Reinforce indentation early: show an error, then fix it.
- Live-code infinite loop → use Ctrl+C to escape (students love this demo).
- Use simple real-world analogies:
  - **Functions** = "machines" with inputs and outputs.
  - **Loops** = "repeating chores until done."
  - **Scope** = "who can see which variable?"
  - **Conditionals** = "decision-making."
- Keep the project **iterative**:
  - Start with just `get_task()`.
  - Add validation.
  - Add scoring.
  - Add bonus/random.

    - Add looping.

Common Pitfalls

- Forgetting to `return` in functions.
- Infinite loops (`while True` without `break`).
- Confusing = with ==.
- Forgetting to cast inputs (`input()` is always a string).
- Forgetting `.strip()` when validating inputs.

Stretch for Fast Learners

- Add "motivation bonus" as a percentage instead of integer.
- Add categories (`work`, `study`, `exercise`) and score differently.
- Use `try/except` for **file saving** (write logs to a file).