

Endpoint Being Tested:

http://127.0.0.1:5000/lecture_summary/<int:lecture_id>

Case: *Successful summary generation*

Request Method: GET

Inputs:

```
{
  "lecture_id": 1
}
```

Expected Output:

HTTP Status Code: 200 and JSON with 'lecture_summary'

Actual Output:

```
HTTP Status Code: 200
JSON: {"lecture_summary": "This introductory lecture to programming using Python emphasizes the practical, hands-on approach to learning. The lecturer argues that learning to program is like learning a language to communicate with a computer. Python is chosen for its widespread use, ease of learning, and power. Unlike older languages, Python allows beginners to start coding almost immediately.\n\nThe lecture uses the analogy of learning to drive: reading about driving is insufficient; one must actually get behind the wheel. Similarly, effective programming learning requires active coding from the very first session. The course will progress incrementally, starting with very simple programs (less than half a page of code) and gradually increasing in complexity. Students are strongly encouraged to follow along by opening a terminal and coding alongside the instructors throughout the course. The initial focus is on understanding the code and its output, with the pace picking up later. No prior computer science knowledge is assumed, only basic English comprehension."}
```

Result: Success

Case: *Transcript not found*

Request Method: GET

Inputs:

```
{  
  "lecture_id": 99999  
}
```

Expected Output:

HTTP Status Code: 404 and error message

Actual Output:

HTTP Status Code: 404
JSON: {"Error": "Transcript not found or empty"}

Result: Success

Case: *Internal server error on summary generation*

Request Method: GET

Inputs:

```
{  
  "lecture_id": 5  
}
```

Expected Output:

HTTP Status Code: 500 and error message

Actual Output:

HTTP Status Code: 200
JSON: {"lecture_summary": "This lecture introduces basic programming concepts using Python. The core idea revolves around creating interactive programs that take user input and respond accordingly.\n\n**1. String Input:** The lecture begins by showing how to take string input from the user (e.g., a name) using `str(input())`, storing it in a variable (e.g., `n`). The program then uses a `print()` statement to display a greeting incorporating the user's input.\n\n**2. Multiple Inputs and Variable Usage:** The program is expanded to take multiple inputs: name and location. The lecturer emphasizes the importance of using distinct variable names (e.g., `n` for name, `p` for place) to avoid overwriting

data. A crucial point is made about data types; strings (`str`) are used for text input, while integers (`int`) would be used for numerical input. An error is deliberately introduced (using an undefined variable) to demonstrate common coding mistakes and the importance of careful variable handling.

3. Combining Inputs and Output:** The program is further refined to generate a more sophisticated output, combining the user's name and location in a personalized greeting. The lecturer points out a minor grammatical issue (a space) and suggests using format specifiers (a topic for future discussion) to address it.

4. Integer Input:** The program is extended to include integer input (e.g., age) using `int(input())`, demonstrating how to handle different data types within the same program. The output is modified to include the user's age.

5. Key Takeaways:** The lecture emphasizes the interactive nature of programming, the importance of correct variable usage and data type handling, and the practical application of `input()` and `print()` statements. The lecturer encourages the learner to explore solutions to minor issues (like removing extra spaces) through online resources. The overall approach is hands-on, showing code examples, executing them, and explaining errors and their solutions in real-time."}

Result: Failed
