

## Task 2

### 1. Task Description:

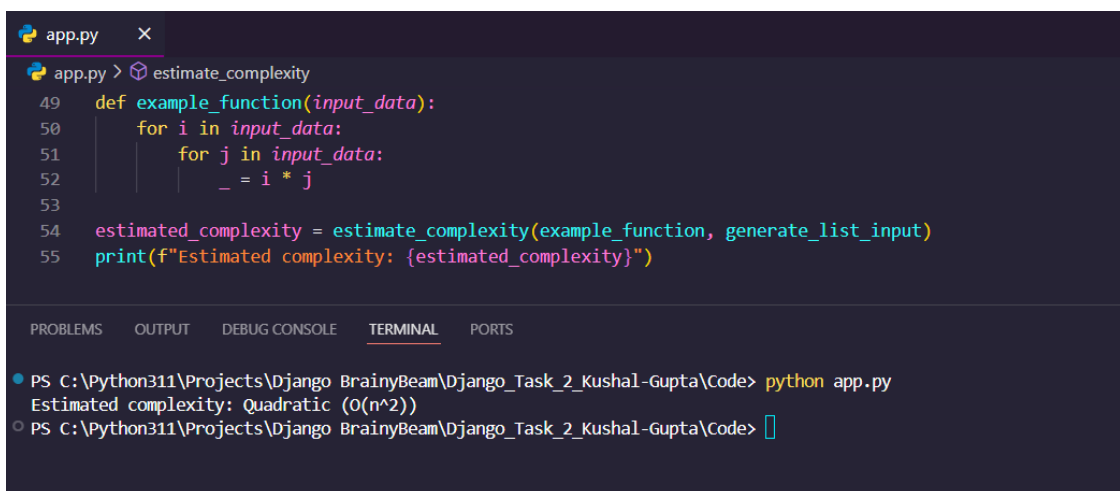
Build a code which can apply to any function and returns their time complexity.

### 2. Task Output Screenshot:



```
app.py X
app.py > estimate_complexity
49 def example_function():
50     return 45 * 45
51
52 estimated_complexity = estimate_complexity(example_function, generate_list_input)
53 print(f"Estimated complexity: {estimated_complexity}")

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Python311\Projects\Django BrainyBeam\Django_Task_2_Kushal-Gupta\Code> python app.py
Estimated complexity: Constant (O(1))
PS C:\Python311\Projects\Django BrainyBeam\Django_Task_2_Kushal-Gupta\Code>
```



```
app.py X
app.py > estimate_complexity
49 def example_function(input_data):
50     for i in input_data:
51         for j in input_data:
52             _ = i * j
53
54 estimated_complexity = estimate_complexity(example_function, generate_list_input)
55 print(f"Estimated complexity: {estimated_complexity}")

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Python311\Projects\Django BrainyBeam\Django_Task_2_Kushal-Gupta\Code> python app.py
Estimated complexity: Quadratic (O(n^2))
PS C:\Python311\Projects\Django BrainyBeam\Django_Task_2_Kushal-Gupta\Code>
```



```
app.py X
app.py > ...
49 def example_function(input_data):
50     for i in input_data:
51         for j in input_data:
52             for k in input_data:
53                 _ = i * j * k
54
55 estimated_complexity = estimate_complexity(example_function, generate_list_input)
56 print(f"Estimated complexity: {estimated_complexity}")

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Python311\Projects\Django BrainyBeam\Django_Task_2_Kushal-Gupta\Code> python app.py
Estimated complexity: Cubic (O(n^3))
PS C:\Python311\Projects\Django BrainyBeam\Django_Task_2_Kushal-Gupta\Code>
```

### 3. Algorithm Used in Task:

The algorithm takes a function and an input data generator as inputs, and it estimates the time complexity of the function by empirically measuring its runtime for different input sizes.

The key steps of the algorithm are:

- 1) Define a set of input sizes to test (e.g. 10, 20, 40, 80, 160, 320).
- 2) For each input size:
  - a. Generate the input data using the provided input generator function.
  - b. Measure the time it takes to execute the target function with the input data.
- 3) Normalize the measured runtimes by dividing them by the runtime of the smallest input size.
- 4) Compare the normalized runtimes to the expected growth rates of various time complexity classes (e.g.  $O(1)$ ,  $O(\log n)$ ,  $O(n)$ ,  $O(n \log n)$ ,  $O(n^2)$ ,  $O(n^3)$ ,  $O(2^n)$ ).
- 5) Determine the time complexity class that best fits the observed runtime growth pattern.

This algorithm provides an empirical way to estimate the time complexity of a function without needing to deeply analyse the function's internal logic. It's a practical approach that can be applied to a wide range of functions and algorithms.