TODO: Identify the Most Attentive Tokens for Each Class in the Test Dataset

The BERT model computes attention scores for each word in every layer, which influence its predictions. After training or fine-tuning the BERT model on the Emotion dataset, the model is expected to focus more on content-specific words relevant to the target class. Your task is to extract the most attentive tokens based on the BERT attention scores. For instance, in the "Happy" class, the most attentive tokens could be "joyful," "wow," or "pleased."

Key Hints:

1. BERT provides an attention score for every token in each layer and for every head, reflecting the importance of one token relative to others in the context
2. For a single sentence, each token will have different attention scores from multiple layers and heads
3. A single token may appear in multiple sentences across the test dataset, and its importance should be aggregated accordingly

Steps to Approach the Task:

1. **Extract Attention Scores**: For each token in the test dataset, collect attention scores from all layers and heads
2. **Aggregate Scores**: Design a method to aggregate attention scores for each token
3. **Rank Tokens**: Identify and rank tokens based on their cumulative attention scores for each class

Implementation Guidance:

- Start with a **pseudo plan** to outline your approach clearly
- Discuss your plan with us
- Use your programming skills

In your code, you need to change the "***BERT_FineTuning***" class and write a function "***class_wise_most_attentive_tokens***" as follows:

```
class BERT_FineTuning(nn.Module):
    def __init__(self, cfg, config_path=None, pretrained=False):
        super().__init__()
        self.cfg = cfg
        self.config = AutoConfig.from_pretrained(cfg.model,
output_hidden_states=True, output_attentions=True) # Changes
```

```
        # The rest of the code

    def forward(self, inputs):
        outputs = self.model(**inputs)
        attentions = outputs.attentions # all layer all head attention
        # The rest of the code
        return logits, attentions
```

```
def class_wise_most_attentive_tokens(test_dataset, class_name,
num_tokens):

      ls_most_attentive_tokens = []

      # Filterout the test sampels that are belong to that class

      # Iterate  through the samples

      # Pass each sample to the model

      # Extract the tokens, predictions and attentions score

      # Find out tokenwise attention score for each token and store them

      # Remember a token can be appear in multiple sentence

      # For all the samples, sort the tokenwise attention score

      return ls_most_attentive_tokens
```