EXP NO: 4

```
for i in range(len(X)):
    in_seq = ' '.join(tokenizer.sequences_to_texts([X[i]])[0].split())
    out_word = tokenizer.index_word[np.argmax(y[i])]

    # Colors
    input_color = '\033[94m'
    arrow_color = '\033[93m'
    output_color = '\033[92m'
    reset = '\033[0m'

    print(f"{input_color}[{in_seq}]{reset} {arrow_color}-> {reset}{output_color}'{out_word}'{reset}")
```

```
[deep] -> 'learning'
[deep learning] -> 'is'
[deep learning is] -> 'amazing'
[deep learning is amazing] -> 'deep'
[deep learning is amazing deep] -> 'learning'
[deep learning is amazing deep learning] -> 'builds'
[deep learning is amazing deep learning builds] -> 'intelligent'
[deep learning is amazing deep learning builds intelligent] -> 'systems'
```

TEST CASES:

```
test_cases = [
    ("Deep learning is", "amazing"),
    ("Deep learning builds", "intelligent"),
    ("Intelligent systems can", "learn"),  # Should fail (word not in training data)
]

# ANSI color codes for colorful output
GREEN = '\033[92m'
RED = '\033[91m'
BLUE = '\033[94m'
YELLOW = '\033[93m'
RESET = '\033[0m'

# Evaluate test cases
for input_seq, expected_word in test_cases:
    # Tokenize and pad the input
    token_list = tokenizer.texts_to_sequences([input_seq])[0]
    token_list = pad_sequences([token_list], maxlen=max_len - 1)

    # Predict the next word
    prediction = model.predict(token_list, verbose=0)
    predicted_index = np.argmax(prediction)
    predicted_word = tokenizer.index_word.get(predicted_index, '?')

    # Compare with expected
    is_correct = (predicted_word.lower() == expected_word.lower())
    result_color = GREEN if is_correct else RED
    result_text = "Y" if is_correct else "N"

    # Print results
    print(f"{BLUE}Input Text:{RESET} {input_seq}")
    print(f"{YELLOW}Predicted Word:{RESET} '{predicted_word}'")
    print(f"{result_color}Correct (Y/N): {result_text}{RESET}\n")
```

OUTPUT:

```
Input Text: Deep learning is
Predicted Word: 'amazing'
Correct (Y/N): Y

Input Text: Deep learning builds
Predicted Word: 'amazing'
Correct (Y/N): N

Input Text: Intelligent systems can
Predicted Word: 'learning'
Correct (Y/N): N
```

ADD-ON:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.utils import pad_sequences, to_categorical

# Example corpus (replace this with your actual cleaned Shakespeare lines)
corpus = [
    "The fool doth think he is wise",
    "But the wise man knows himself to be a fool"
]

# Tokenization
tokenizer = Tokenizer()
tokenizer.fit_on_texts(corpus)
total_words = len(tokenizer.word_index) + 1

# Generate input sequences using n-grams
input_sequences = []
for line in corpus:
    token_list = tokenizer.texts_to_sequences([line])[0]
    for i in range(1, len(token_list)):
        n_gram_sequence = token_list[:i + 1]
        input_sequences.append(n_gram_sequence)

# Pad sequences
max_len = max(len(seq) for seq in input_sequences)
input_sequences = pad_sequences(input_sequences, maxlen=max_len, padding='pre')

# Create predictors and label
X = input_sequences[:, :-1]
y = input_sequences[:, -1]
y = to_categorical(y, num_classes=total_words)

# Define the model
model = Sequential([
    Embedding(input_dim=total_words, output_dim=100, input_length=max_len - 1),
    LSTM(150),
    Dense(total_words, activation='softmax')
])

# Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the model
model.fit(X, y, epochs=50, verbose=1)
```

OUTPUT:

```
Epoch 24/50
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 69ms/step - accuracy: 0.4000 - loss: 2.1334
Epoch 25/50
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - accuracy: 0.4000 - loss: 2.0743
Epoch 26/50
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - accuracy: 0.4000 - loss: 2.0075
Epoch 27/50
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 62ms/step - accuracy: 0.4000 - loss: 1.9317
Epoch 28/50
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - accuracy: 0.4000 - loss: 1.8535
Epoch 29/50
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step - accuracy: 0.4000 - loss: 1.7794
Epoch 30/50
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step - accuracy: 0.3333 - loss: 1.7006
Epoch 31/50
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - accuracy: 0.3333 - loss: 1.6197
Epoch 32/50
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - accuracy: 0.4000 - loss: 1.5338
```