# Student HELPER APP

Bilingual Educational Assistant

Nojood Alnahdi
Shahad Albalawi

## MOOD TRACKER APPLICATION

allows users to log their daily experiences in various languages, classify their mood based on their text entries, and receive motivational messages. project utilizes a combination of Hugging Face pipelines, OpenAI GPT-3.5-turbo API, and Gradio

## MODELS AND PIPELINES

### 1 TEXT CLASSIFICATION:

Tpowered by a pre-trained RoBERTa model (SamLowe/roberta-base-go_emotions), which classifies input text into one of several emotional labels.

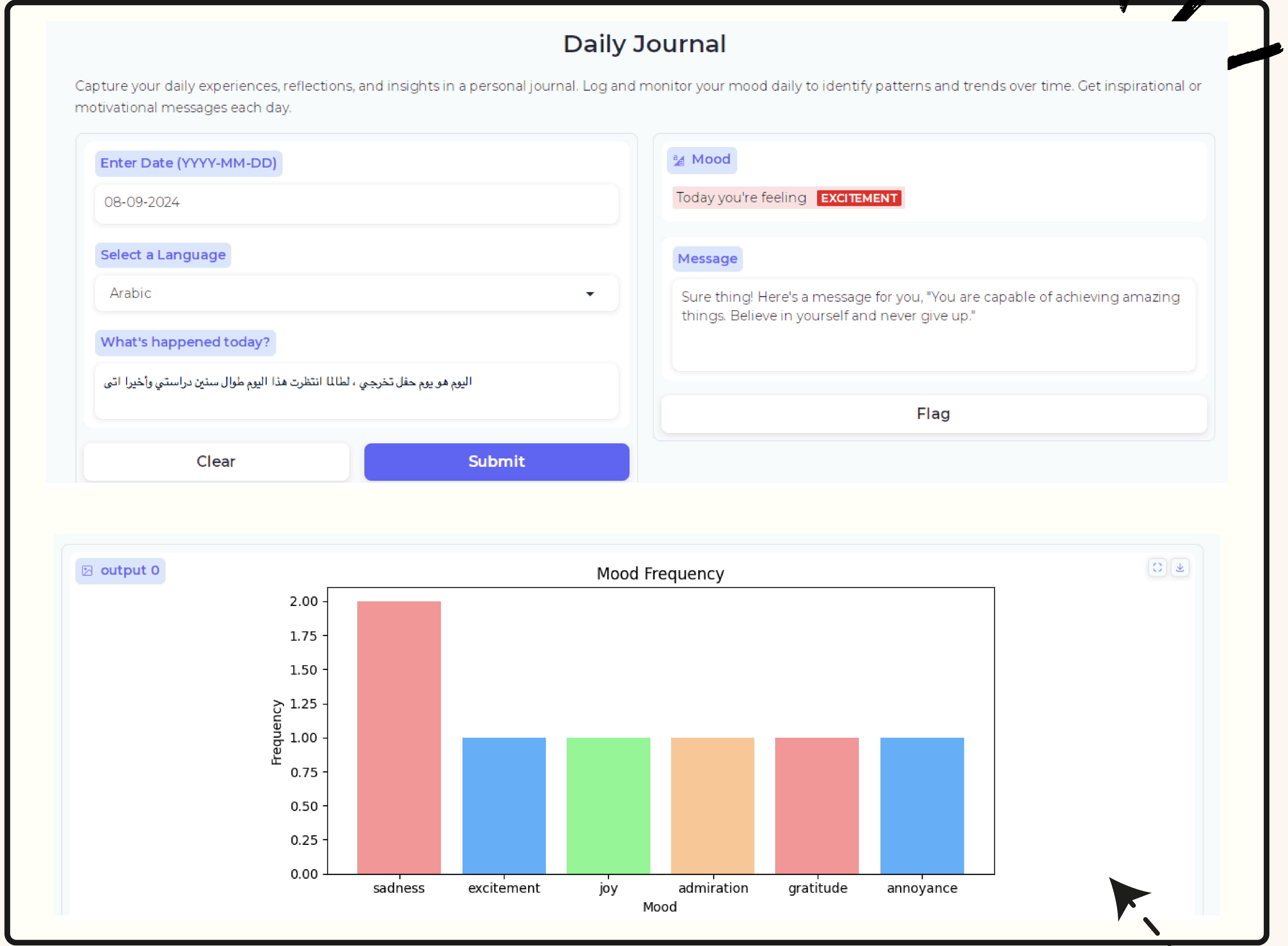
### 2 TRANSLATION:

The facebook/nllb-200-distilled-600M model translates user inputs from non-English languages to English.

### 3 OPENAI GPT-3.5 API:

After the mood classification, the OpenAI API generates a message based on the detected mood

# Previous Work



Daily Journal

Capture your daily experiences, reflections, and insights in a personal journal. Log and monitor your mood daily to identify patterns and trends over time. Get inspirational or motivational messages each day.

Enter Date (YYYY-MM-DD): 08-09-2024

Select a Language: Arabic

What's happened today?: اليوم هو يوم حفل تخرجي ، لطالما انتظرت هذا اليوم طوال سنين دراستي وأخيرا اتى

Mood: Today you're feeling EXCITEMENT

Message: Sure thing! Here's a message for you, "You are capable of achieving amazing things. Believe in yourself and never give up."

## Description

The Student Helper App helps students manage and review educational content by offering transcription, summarization, translation, question generation, and interactive Q&A in both Arabic and English.
Students can upload audio files, video files, or provide YouTube links to convert speech into text, enabling them to review and engage with educational materials effectively.

## Project Objectives

1. Assisting students in managing and reviewing educational content.
2. Supporting bilingual functionality (Arabic and English).
3. Leveraging machine learning models for transcription, summarization, translation, Q&A generation, and audio conversion.

# Models and Pipelines

## Transcription

- Using OpenAI Whisper for accurate transcription, to converts audio/video content into text(audio-to-text )

## Summarization

- Leveraging BART (facebook/bart-large-cnn) to summarize transcribed content effectively.

## Translation

- Utilizing NLLB-200 (facebook/nllb-200-distilled-600M) for high-quality bilingual (Arabic and English) translation.

## Q&A Generation

- Implementing (valhalla/t5-small-qg-prepend) for generating educational Q&A from the transcribed content.

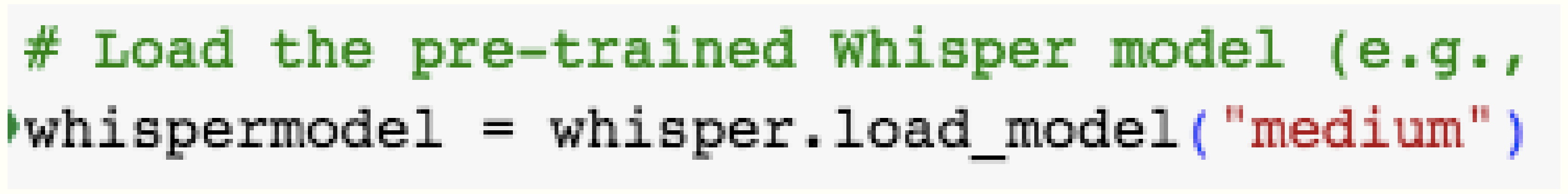
## Interactive Q&A

- Employing RoBERTa(deepset/roberta-base-squad2) to provide answers to student questions based on the content.

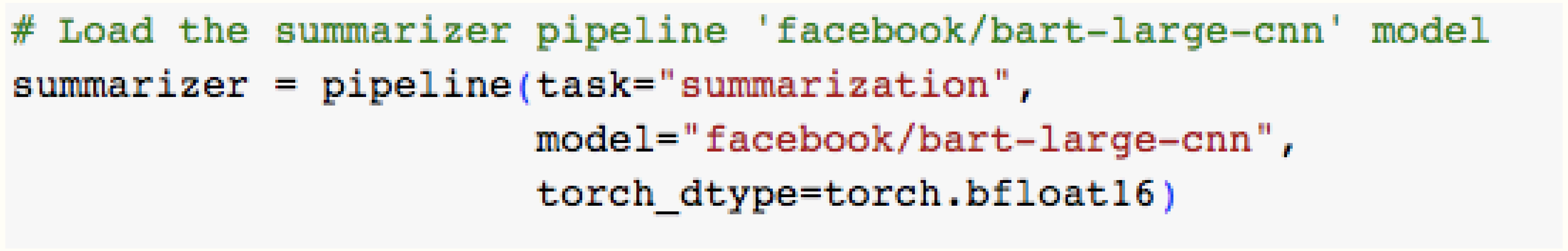
## Text-to-Speech

- Using gTTS for converting summarized text into audio in the user's preferred language, making it accessible in audio format

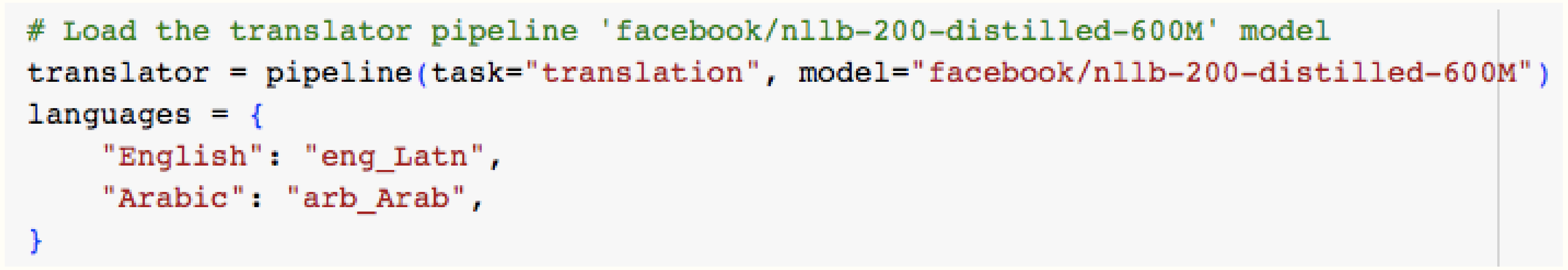
# Models and Pipelines



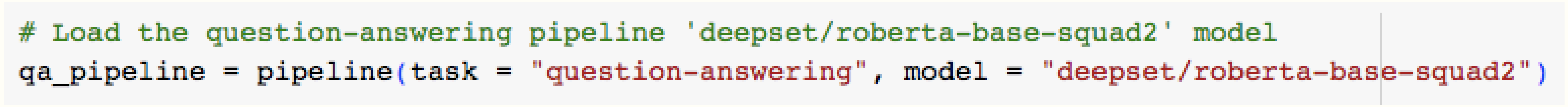
```python
# Load the pre-trained Whisper model (e.g.,
whispermodel = whisper.load_model("medium")
```



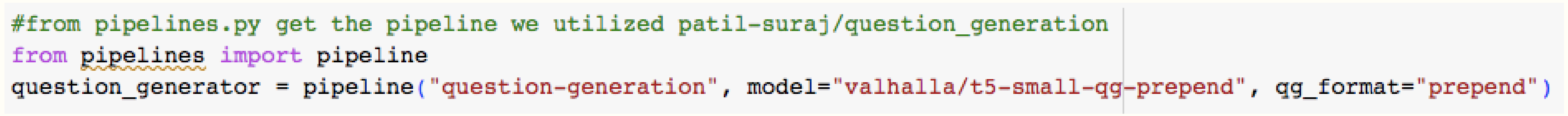
```python
# Load the summarizer pipeline 'facebook/bart-large-cnn' model
summarizer = pipeline(task="summarization",
                      model="facebook/bart-large-cnn",
                      torch_dtype=torch.bfloat16)
```



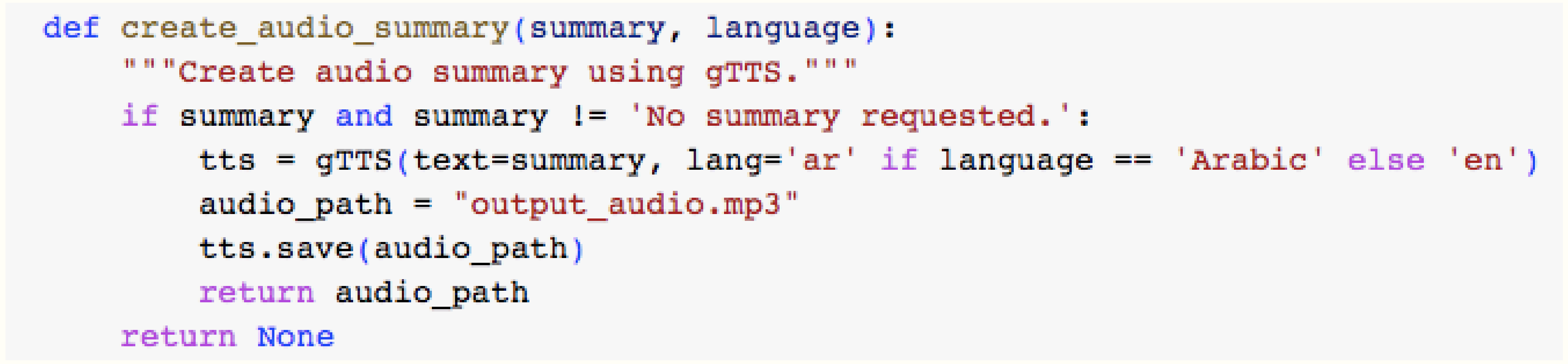
```python
# Load the translator pipeline 'facebook/nllb-200-distilled-600M' model
translator = pipeline(task="translation", model="facebook/nllb-200-distilled-600M")
languages = {
    "English": "eng_Latn",
    "Arabic": "arb_Arab",
}
```



```python
# Load the question-answering pipeline 'deepset/roberta-base-squad2' model
qa_pipeline = pipeline(task = "question-answering", model = "deepset/roberta-base-squad2")
```

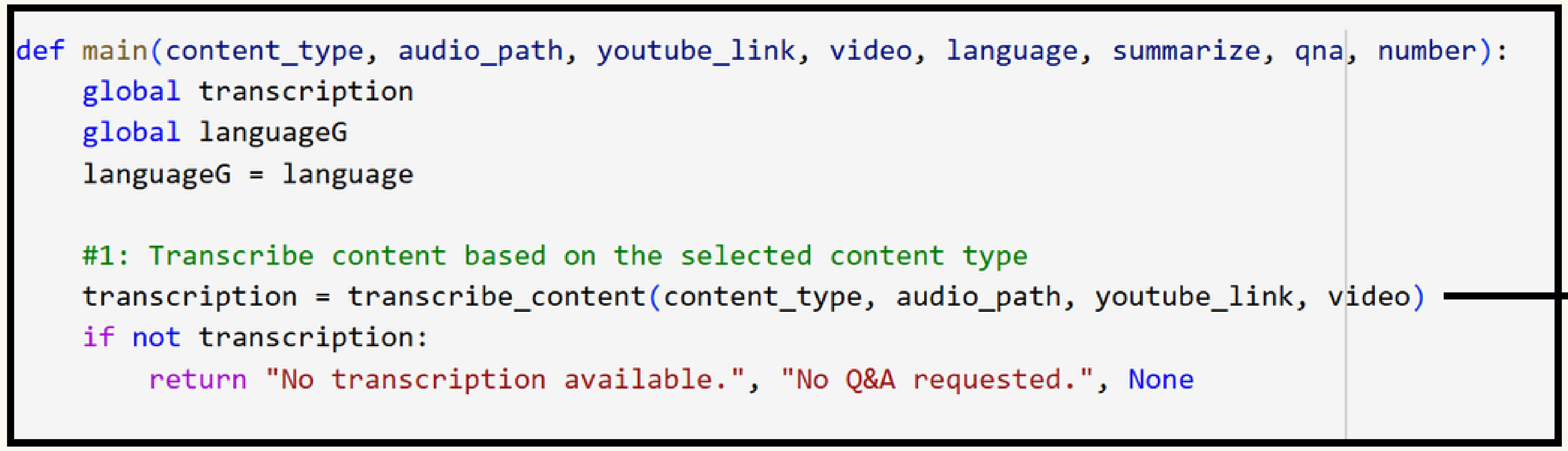


```python
#from pipelines.py get the pipeline we utilized patil-suraj/question_generation
from pipelines import pipeline
question_generator = pipeline("question-generation", model="valhalla/t5-small-qg-prepend", qg_format="prepend")
```

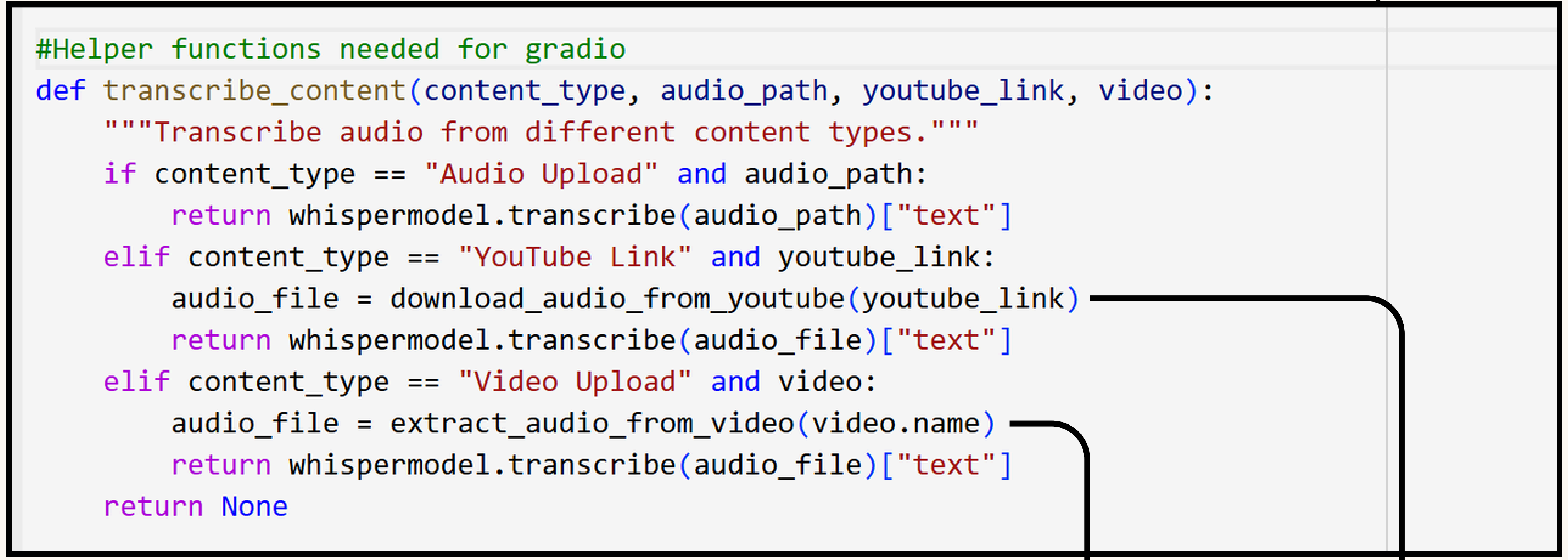


```python
def create_audio_summary(summary, language):
    """Create audio summary using gTTS."""
    if summary and summary != 'No summary requested.':
        tts = gTTS(text=summary, lang='ar' if language == 'Arabic' else 'en')
        audio_path = "output_audio.mp3"
        tts.save(audio_path)
        return audio_path
    return None
```

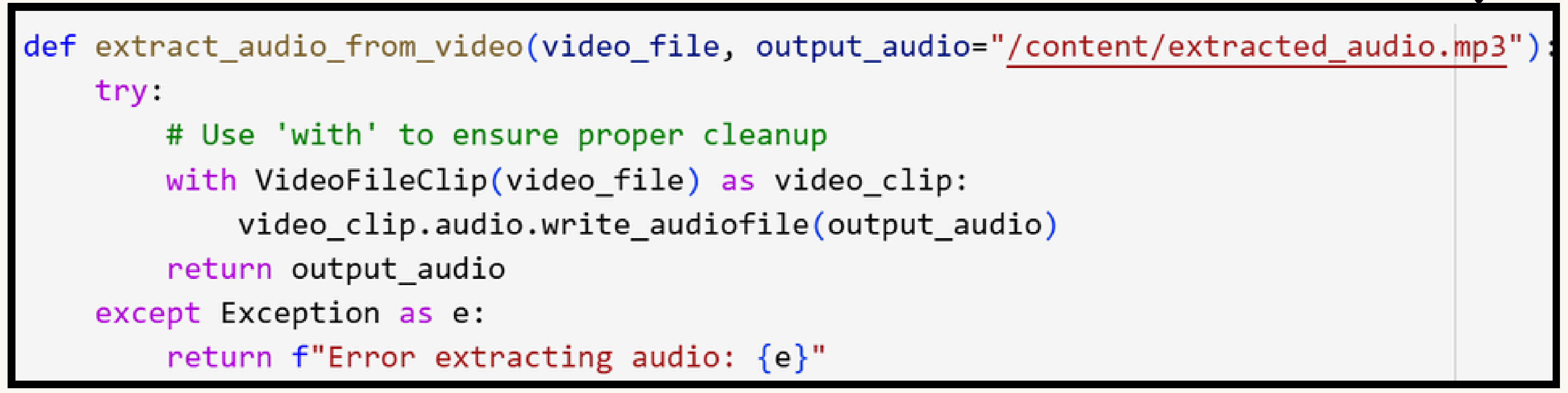
# FUNCTIONS



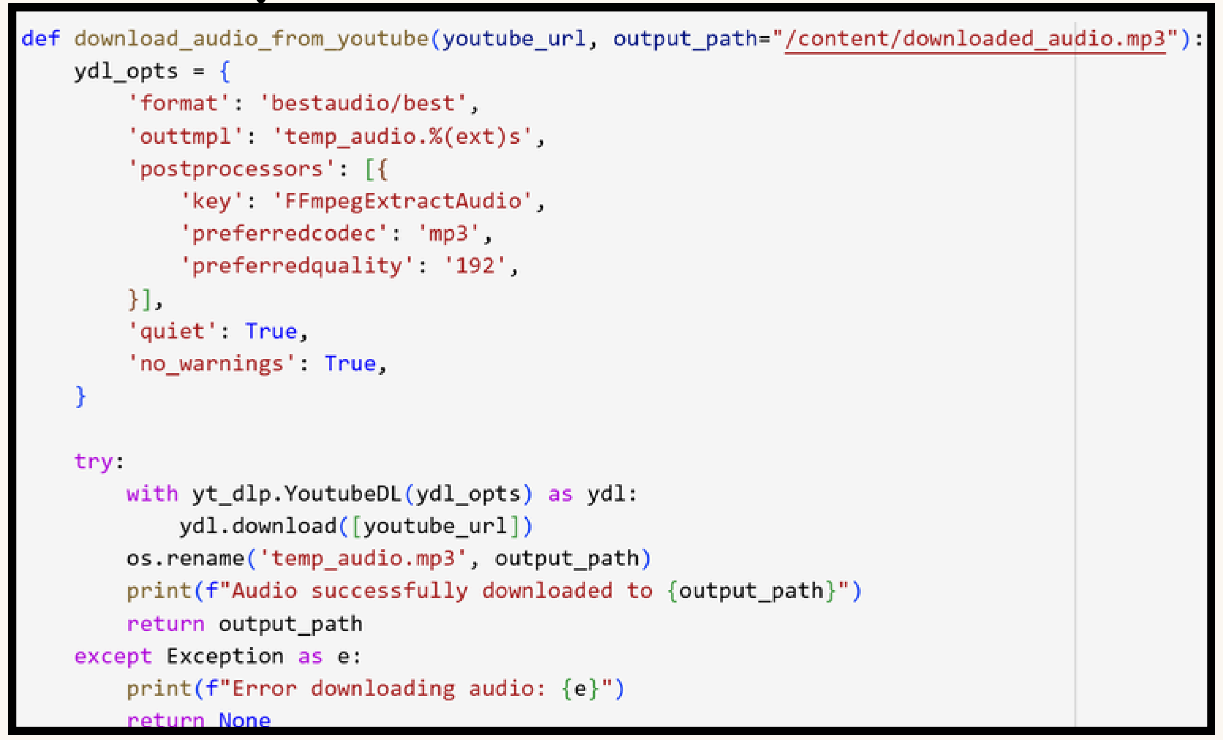
```python
def main(content_type, audio_path, youtube_link, video, language, summarize, qna, number):
    global transcription
    global languageG
    languageG = language

    #1: Transcribe content based on the selected content type
    transcription = transcribe_content(content_type, audio_path, youtube_link, video)
    if not transcription:
        return "No transcription available.", "No Q&A requested.", None
```



```python
#Helper functions needed for gradio
def transcribe_content(content_type, audio_path, youtube_link, video):
    """Transcribe audio from different content types."""
    if content_type == "Audio Upload" and audio_path:
        return whispermodel.transcribe(audio_path)["text"]
    elif content_type == "YouTube Link" and youtube_link:
        audio_file = download_audio_from_youtube(youtube_link)
        return whispermodel.transcribe(audio_file)["text"]
    elif content_type == "Video Upload" and video:
        audio_file = extract_audio_from_video(video.name)
        return whispermodel.transcribe(audio_file)["text"]
    return None
```

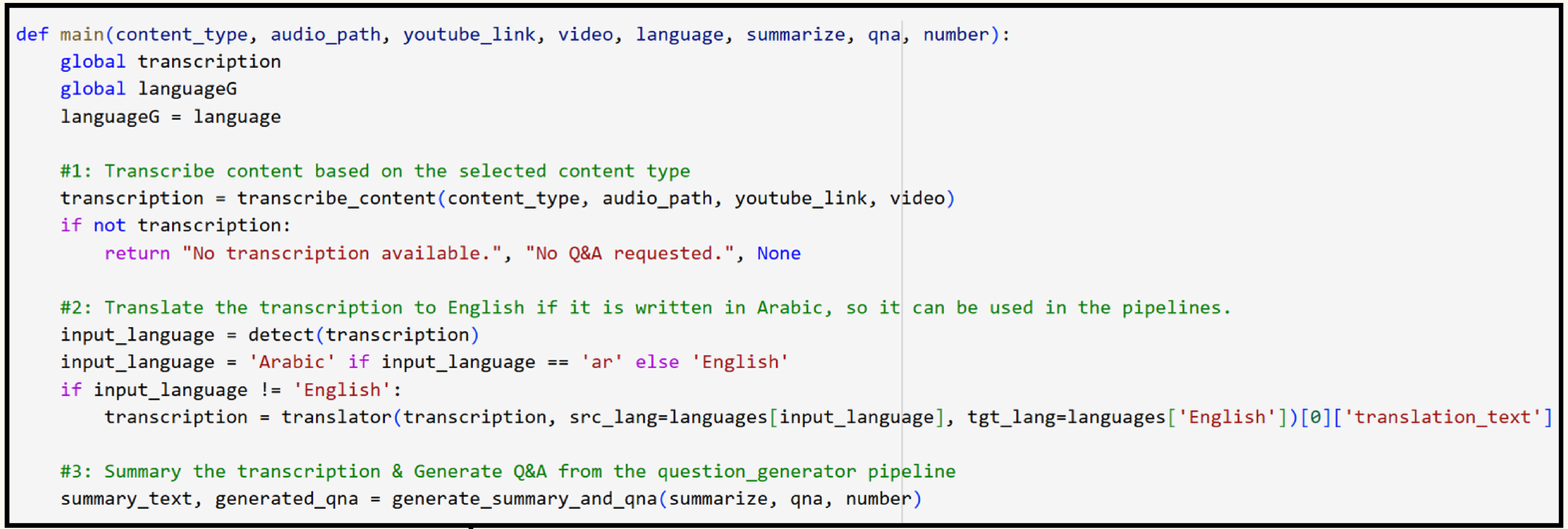


```python
def extract_audio_from_video(video_file, output_audio="/content/extracted_audio.mp3"):
    try:
        # Use 'with' to ensure proper cleanup
        with VideoFileClip(video_file) as video_clip:
            video_clip.audio.write_audiofile(output_audio)
        return output_audio
    except Exception as e:
        return f"Error extracting audio: {e}"
```

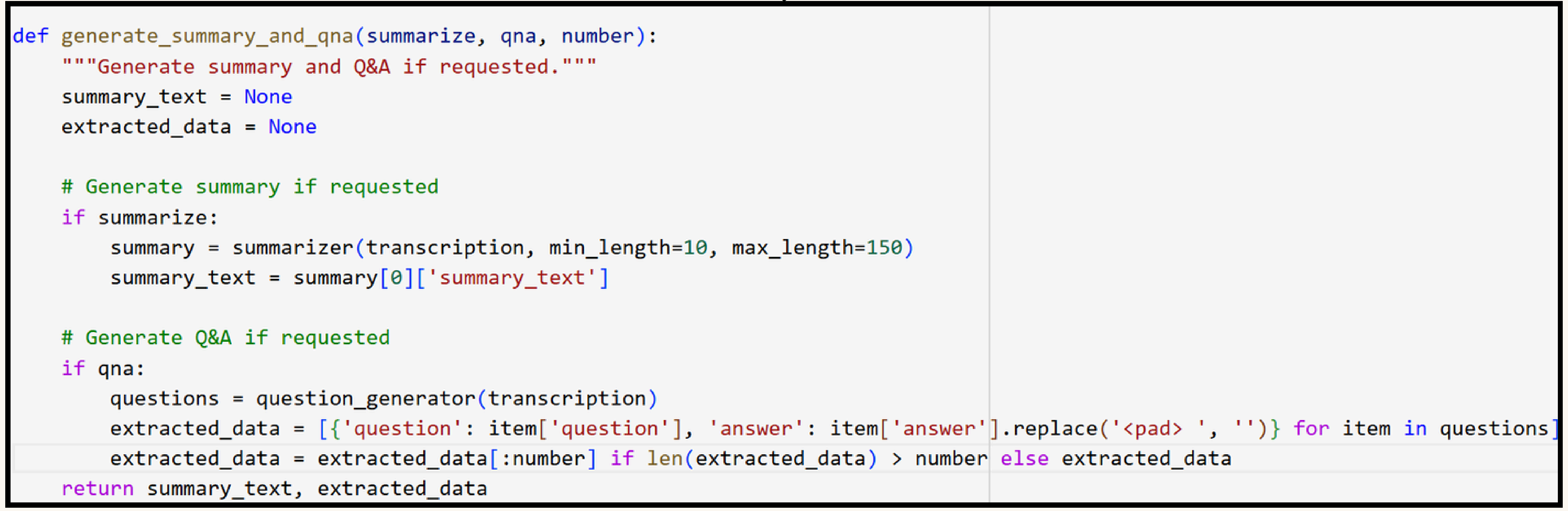


```python
def download_audio_from_youtube(youtube_url, output_path="/content/downloaded_audio.mp3"):
    ydl_opts = {
        'format': 'bestaudio/best',
        'outtmpl': 'temp_audio.%(ext)s',
        'postprocessors': [{
            'key': 'FFmpegExtractAudio',
            'preferredcodec': 'mp3',
            'preferredquality': '192',
        }],
        'quiet': True,
        'no_warnings': True,
    }

    try:
        with yt_dlp.YoutubeDL(ydl_opts) as ydl:
            ydl.download([youtube_url])
        os.rename('temp_audio.mp3', output_path)
        print(f"Audio successfully downloaded to {output_path}")
        return output_path
    except Exception as e:
        print(f"Error downloading audio: {e}")
        return None
```

## FUNCTIONS

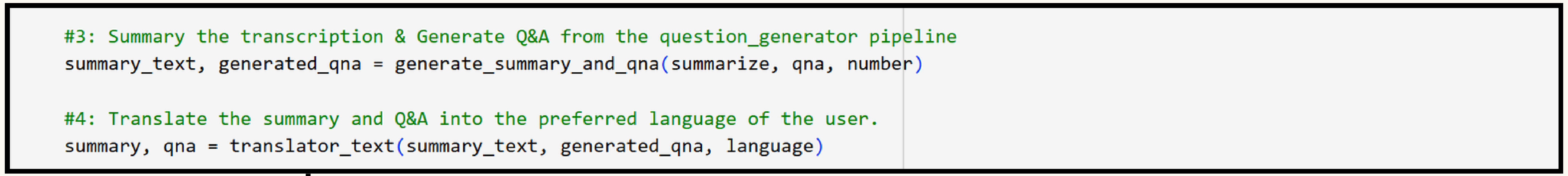


```python
def main(content_type, audio_path, youtube_link, video, language, summarize, qna, number):
    global transcription
    global languageG
    languageG = language

    #1: Transcribe content based on the selected content type
    transcription = transcribe_content(content_type, audio_path, youtube_link, video)
    if not transcription:
        return "No transcription available.", "No Q&A requested.", None

    #2: Translate the transcription to English if it is written in Arabic, so it can be used in the pipelines.
    input_language = detect(transcription)
    input_language = 'Arabic' if input_language == 'ar' else 'English'
    if input_language != 'English':
        transcription = translator(transcription, src_lang=languages[input_language], tgt_lang=languages['English'])[0]['translation_text']

    #3: Summary the transcription & Generate Q&A from the question_generator pipeline
    summary_text, generated_qna = generate_summary_and_qna(summarize, qna, number)
```

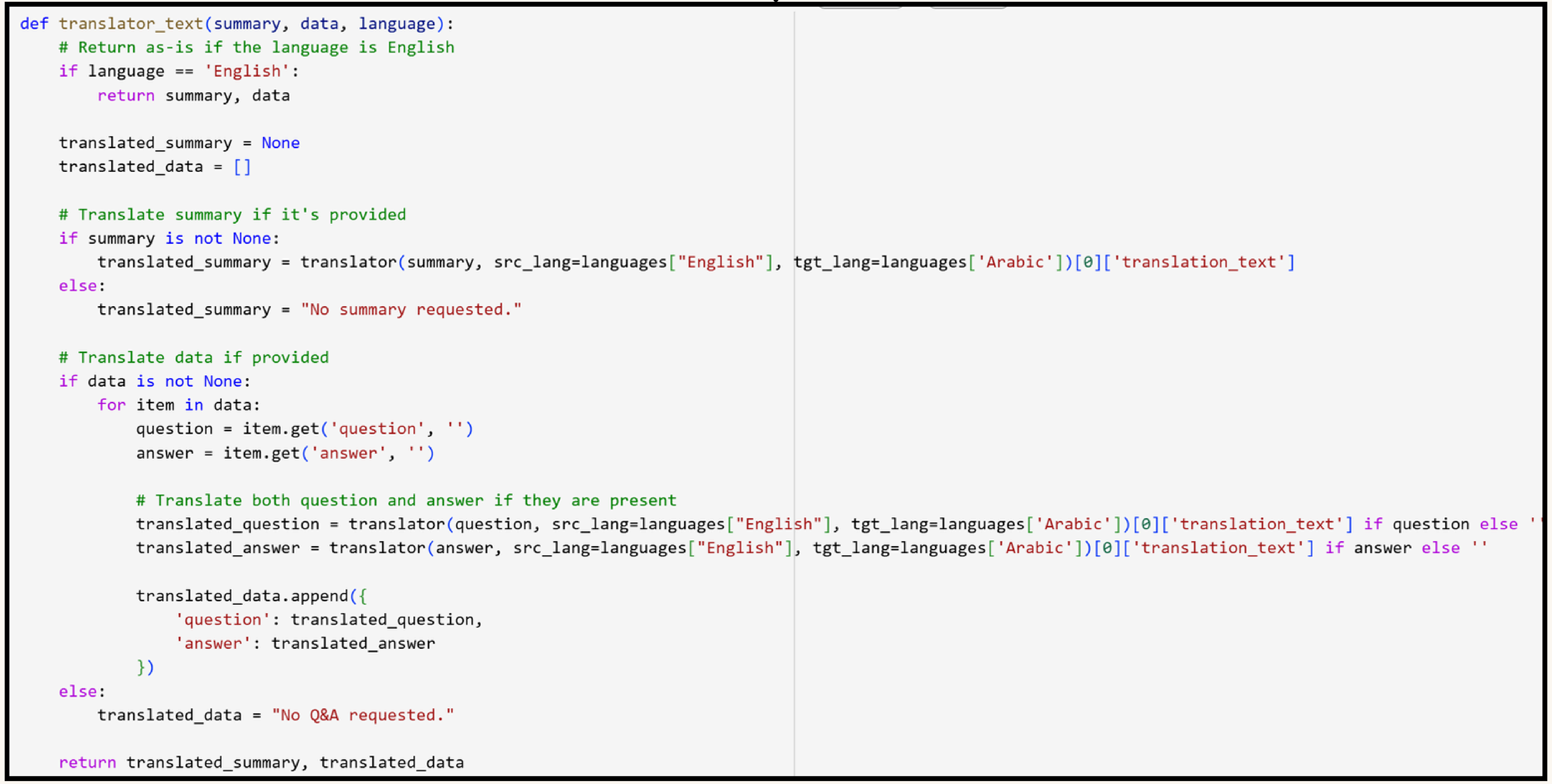


```python
def generate_summary_and_qna(summarize, qna, number):
    """Generate summary and Q&A if requested."""
    summary_text = None
    extracted_data = None

    # Generate summary if requested
    if summarize:
        summary = summarizer(transcription, min_length=10, max_length=150)
        summary_text = summary[0]['summary_text']

    # Generate Q&A if requested
    if qna:
        questions = question_generator(transcription)
        extracted_data = [{'question': item['question'], 'answer': item['answer'].replace('<pad> ', '')} for item in questions]
        extracted_data = extracted_data[:number] if len(extracted_data) > number else extracted_data
    return summary_text, extracted_data
```

## FUNCTIONS



```python
#3: Summary the transcription & Generate Q&A from the question_generator pipeline
summary_text, generated_qna = generate_summary_and_qna(summarize, qna, number)

#4: Translate the summary and Q&A into the preferred language of the user.
summary, qna = translator_text(summary_text, generated_qna, language)
```



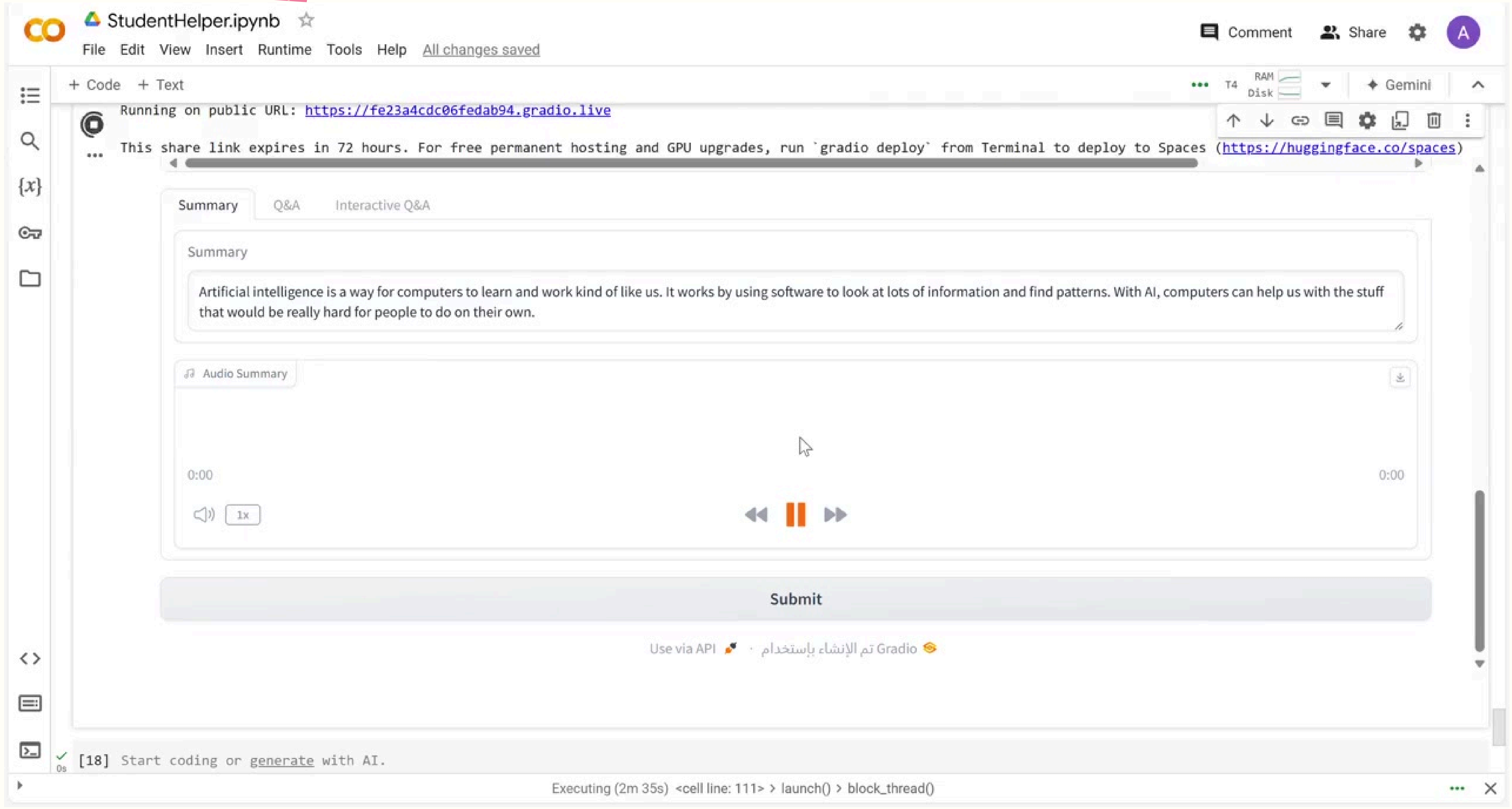
```python
def translator_text(summary, data, language):
    # Return as-is if the language is English
    if language == 'English':
        return summary, data

    translated_summary = None
    translated_data = []

    # Translate summary if it's provided
    if summary is not None:
        translated_summary = translator(summary, src_lang=languages["English"], tgt_lang=languages['Arabic'])[0]['translation_text']
    else:
        translated_summary = "No summary requested."

    # Translate data if provided
    if data is not None:
        for item in data:
            question = item.get('question', '')
            answer = item.get('answer', '')

            # Translate both question and answer if they are present
            translated_question = translator(question, src_lang=languages["English"], tgt_lang=languages['Arabic'])[0]['translation_text'] if question else '
            translated_answer = translator(answer, src_lang=languages["English"], tgt_lang=languages['Arabic'])[0]['translation_text'] if answer else ''

            translated_data.append({
                'question': translated_question,
                'answer': translated_answer
            })
    else:
        translated_data = "No Q&A requested."

    return translated_summary, translated_data
```

# FUNCTIONS

```python
#5: Generate audio from the summary to be in the user's preferred language.
audio_path = create_audio_summary(summary, language)

#6: Prepare Q&A output
qna_output = (
    "\n\n".join(
        f"**Question:** {item['question']}\n**Answer:** {item['answer']}"
        if language == "English"
        else f"**السؤال:** {item['question']}\n**الجواب:** {item['answer']}"
        for item in qna
    ) if qna else "No Q&A requested."
)

return summary, qna_output, audio_path
```

```python
def create_audio_summary(summary, language):
    """Create audio summary using gTTS."""
    if summary and summary != 'No summary requested.':
        tts = gTTS(text=summary, lang='ar' if language == 'Arabic' else 'en')
        audio_path = "output_audio.mp3"
        tts.save(audio_path)
        return audio_path
    return None
```

# Results



```
Running on public URL: https://fe23a4cdc06fedab94.gradio.live

This share link expires in 72 hours. For free permanent hosting and GPU upgrades, run `gradio deploy` from Terminal to deploy to Spaces (https://huggingface.co/spaces)
```

Summary

Artificial intelligence is a way for computers to learn and work kind of like us. It works by using software to look at lots of information and find patterns. With AI, computers can help us with the stuff that would be really hard for people to do on their own.

```
[18] Start coding or generate with AI.
```

Executing (2m 35s) <cell line: 111> > launch() > block_thread()

## ADDRESS

- GitHub repository link
- Hugging Face space link

## CONCLUSIONS

The Student Helper App simplifies the process of reviewing educational content by providing transcription, summarization, Q&A generation, and interactive Q&A in both Arabic and English. Its bilingual support ensures accessibility for a diverse audience.

Future enhancements could include expanding to more languages and improving Q&A capabilities, making the app even more versatile for students.

THANK YOU!