**Pytube package:**

PyTube is a Python package for downloading videos from YouTube. It provides a simple and easy-to-use interface for downloading videos in various resolutions and formats. With PyTube, you can download YouTube videos by specifying the video URL or video ID.

**path = yt.streams.filter(file\_extension="mp4").get\_by\_resolution("360p").download(outpath)**

This line of code is using PyTube to download a YouTube video stream in MP4 format with a resolution of 360p and saving it to the specified output path outpath.

* yt is a YouTube object that has been instantiated with a video URL or ID.
* streams is a list of Stream objects that are available for download from the video.
* filter(file\_extension="mp4") filters the list of streams to only include those that have an MP4 file extension.
* get\_by\_resolution("360p") selects a stream with a resolution of 360p from the filtered list.
* download(outpath) downloads the selected stream and saves it to the specified output path.

**moviepy.editor package**

The moviepy.editor package is a module in the MoviePy library that provides a high-level interface for creating and manipulating video clips. This package includes the VideoFileClip, AudioFileClip, and CompositeVideoClip classes, among others, that make it easy to work with video and audio files in Python.

**clip = mp.VideoFileClip(path)**

This line of code is using the VideoFileClip class from the MoviePy package to create a new video clip object from a video file located at the specified path.

* mp refers to the moviepy module, which is a Python library for working with video files.
* VideoFileClip(path) is a constructor for the VideoFileClip class that creates a new video clip object from the video file at the specified path.
* The resulting clip object can then be used to perform various operations on the video, such as cutting, trimming, and adding effects.

**clip.audio.write\_audiofile("test.wav",codec='pcm\_s16le')**

This line of code is using the write\_audiofile method of a VideoFileClip object to export the audio of the clip to a WAV file with the specified file name "test.wav". The codec parameter is set to "pcm\_s16le", which specifies the audio codec to use for the output file.

* clip is a VideoFileClip or AudioFileClip object that has been instantiated from a video or audio file.
* clip.audio is an AudioFileClip object that represents the audio track of the clip.
* write\_audiofile is a method of the AudioFileClip object that exports the audio track to a file in a specified format and codec.
* The first argument of write\_audiofile is the file name and path of the output file.
* The codec parameter specifies the audio codec to use for the output file. In this case, "pcm\_s16le" specifies a 16-bit PCM format with little-endian byte order. The pcm\_s16le codec is often used for recording, editing, and storing high-quality audio, such as music or voice recordings.

**wav file format**

The WAV (Waveform Audio File Format) is a standard audio file format used for storing digital audio data on a computer.

**audio = AudioSegment.from\_file(path)**

the from\_file() method of the AudioSegment provides a convenient way to read audio data from a file and work with it in Python using the pydub library.

tokenizer = PegasusTokenizer.from\_pretrained("google/pegasus-wikihow")

The PegasusTokenizer class is a part of the Hugging Face transformers library, which provides a high-level interface for working with pre-trained language models. Specifically, PegasusTokenizer is a tokenizer that is designed to be used with the Pegasus model architecture, which is a state-of-the-art language model that is pre-trained on large amounts of text data.

The from\_pretrained() method of the PegasusTokenizer class is used to create a tokenizer object that is pre-trained on a specific model. In this case, the model being used is "google/pegasus-wikihow", which is a pre-trained Pegasus model that has been fine-tuned on a large corpus of text from the WikiHow website.

When you call from\_pretrained() on a tokenizer, Hugging Face downloads the pre-trained tokenizer files from the specified model and creates a tokenizer object that is ready to use. The tokenizer can then be used to tokenize input text, breaking it down into individual tokens that can be fed into the Pegasus model for processing.

**model = PegasusForConditionalGeneration.from\_pretrained("google/pegasus-wikihow")**

The PegasusForConditionalGeneration class is a part of the Hugging Face transformers library, which provides a high-level interface for working with pre-trained language models. Specifically, PegasusForConditionalGeneration is a class that implements the Pegasus model architecture, which is a state-of-the-art language model that is pre-trained on large amounts of text data.The PegasusForConditionalGeneration class is designed for conditional generation tasks, where the model is given a prompt or input sequence and asked to generate a new sequence of text that follows the input.

**token = tokenizer(main\_listText, truncation = True, padding = "longest", return\_tensors = "pt")**

The tokenizer() method of a tokenizer object, in this case a PegasusTokenizer object, is used to tokenize input text, breaking it down into individual tokens that can be fed into a language model like Pegasus for processing.

In this specific line of code, main\_listText is a list of strings containing the input text to be tokenized. The truncation parameter is set to True, which means that the tokenizer will truncate any input text that is longer than the maximum sequence length allowed by the Pegasus model. The padding parameter is set to "longest", which means that any shorter input texts will be padded with zeros to match the length of the longest input text. The return\_tensors parameter is set to "pt", which means that the output tokens will be returned in a PyTorch tensor format

**summary = model.generate(\*\*token)**

The generate() method is a method of the PegasusForConditionalGeneration class, which is a part of the Hugging Face transformers library. This method generates new text based on the input sequence provided to the model.

In this specific line of code, model is an instance of the PegasusForConditionalGeneration class that has been pre-trained on a large corpus of text data. The \*\*token syntax is used to pass in the output of the tokenizer() method from the previous line of code as keyword arguments to the generate() method. This provides the input sequence to the model, which is used to generate a new sequence of text.

The generate() method returns a PyTorch tensor containing the integer IDs of the tokens that make up the generated text.

**summaryText += tokenizer.decode(summary[i])**

These IDs can be converted back into human-readable text using the decode() method of the tokenizer object.

**Flask code**

This code creates a Flask application object named "app". It defines two routes:

The first route, "/", renders the "index.html" template when a GET request is made to the root URL.

The second route, "/video", accepts both GET and POST requests. When a POST request is made, the function "get\_video\_url()" is executed. This function retrieves the value of the "url" field from the form submitted in the POST request, downloads the video from the URL, converts the audio to text, generates a summary of the text, and renders the "result.html" template with the video URL, generated text, and summary as parameters.

Finally, the code starts the Flask application with the "app.run()" function. The "debug=True" parameter enables debugging mode, which provides more detailed error messages in case of a runtime error.