**Problem Statement –** Frame a model that canbe used to expand the text to make it longer by adding more context.

Input - A sentence

Output – An expanded text with more details and content

For Example –

If you provide a simple sentence for NLP as input, like-

Input - NLP is Natural Language Processing.

It should give a complete paragraph with details about NLP.

Output - NLP is a branch of artificial intelligence that deals with understanding and manipulating human language. It is used to interact with computers in natural language, such as English, Spanish, and other languages. NLP techniques are used to analyze text, extracting relevant information from it, and understanding what a user means when they type something. This is done by applying a set of algorithms to the text, which can be used to identify patterns, categorize words, and generate responses. Common applications of NLP include machine translation, text summarization, question answering, text classification, sentiment analysis, and automated dialogue systems.

Approach-1

This script uses the transformers library to load a pre-trained language model, preprocess a sentence, and generate more text based on the preprocessed sentence using top-p sampling. Specifically, it uses the EleutherAI/gpt-neo-2.7B model, which is a very large language model with 2.7 billion parameters.

The script first imports the necessary libraries, including torch and transformers. It then loads the pre-trained language model and tokenizer using the AutoTokenizer and AutoModelWithLMHead classes from transformers. The sentence to expand is defined as a string variable.

The sentence is then preprocessed using the tokenizer's encode method, which returns a tensor of input IDs. An attention mask is also defined using a tensor of ones with the same shape as the input IDs tensor. Finally, the pad token ID is retrieved from the tokenizer.

The model's generate method is then called with the preprocessed input IDs, attention mask, and pad token ID. The max\_length parameter is set to 100, which specifies the maximum length of the generated output. The do\_sample parameter is set to True to enable sampling of the output, and top\_p and top\_k parameters are set to 0.95 and 50, respectively, to control the quality and diversity of the generated output.

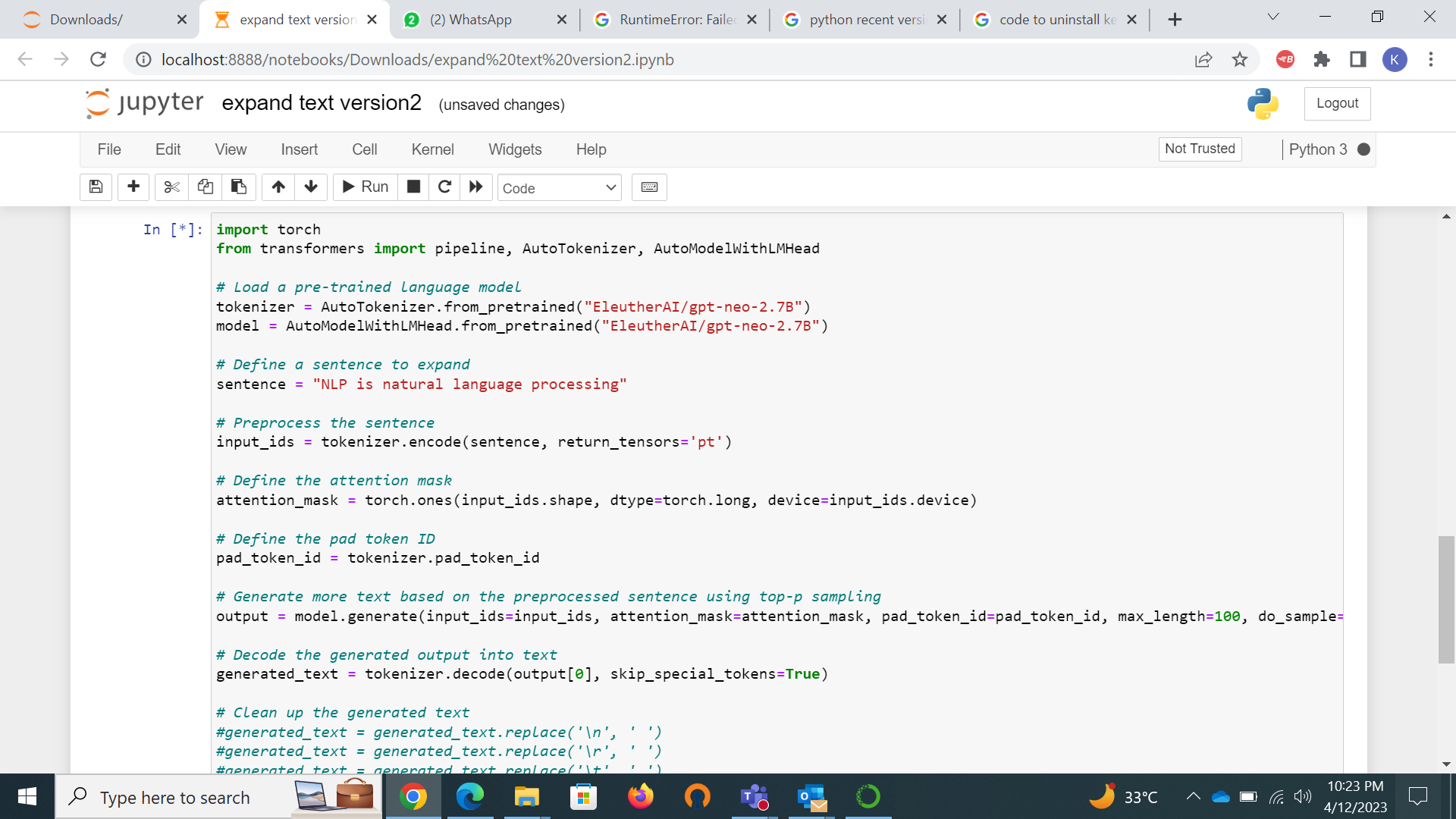
The output of the generate method is a tensor of generated token IDs, which is decoded into text using the tokenizer's decode method, with special tokens skipped. The generated text is then printed to the console.

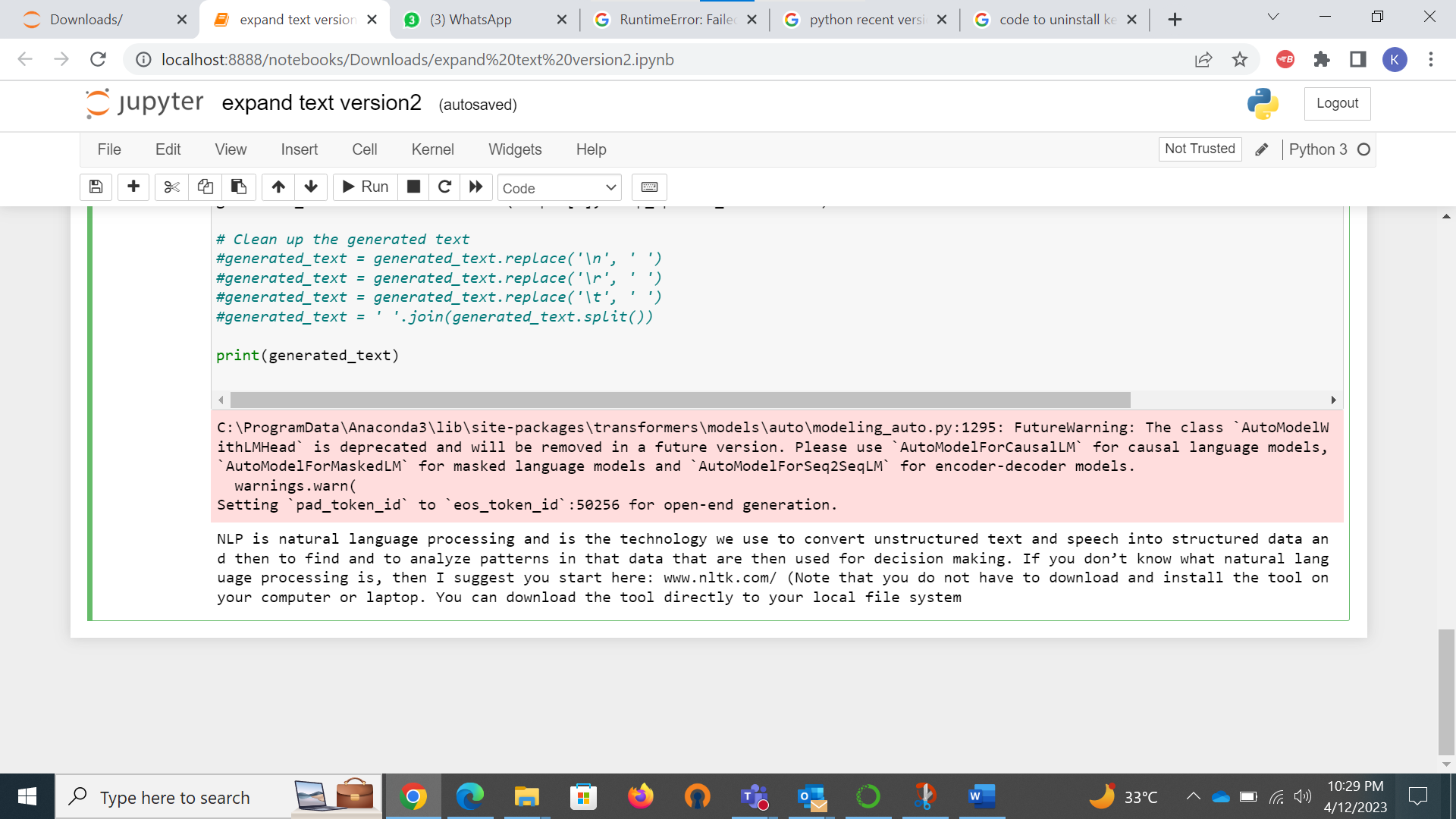
Overall, this script demonstrates how to use a pre-trained language model to generate new text based on a given input sentence.

Algorithm-1

This is a Python script that uses the transformers library to generate more text based on a given input sentence using a pre-trained language model. Here is a step-by-step breakdown of what the script does:

1. First, it imports the necessary modules: torch, which is a PyTorch library for tensor operations, and the transformers library, which provides an easy-to-use interface for working with pre-trained language models.
2. It loads a pre-trained language model called EleutherAI/gpt-neo-2.7B using the AutoTokenizer and AutoModelWithLMHead classes from the transformers library. This model has been pre-trained on a massive corpus of text data and can generate high-quality natural language text.
3. It defines a sentence to expand, which is "ricky ponting was one of the best cricketer of all time".
4. It preprocesses the sentence using the tokenizer.encode() method, which converts the text into a sequence of token IDs that the language model can process.
5. It defines an attention mask with the same shape as the input tensor, which tells the language model which tokens to attend to and which ones to ignore. In this case, it sets all the values to 1, indicating that all tokens should be attended to.
6. It defines the pad token ID using the tokenizer.pad\_token\_id attribute, which is used to pad the input sequence to the same length as the maximum sequence length the model was trained on.
7. It generates more text based on the preprocessed sentence using the model.generate() method. This method takes the preprocessed input sequence, the attention mask, the pad token ID, and several other arguments, and generates a new sequence of tokens using the language model. In this case, it uses top-p sampling, which means that it selects the next token based on the probability distribution of the top tokens whose cumulative probability mass is at most 0.95. It also uses top-k filtering, which means that it only considers the top 50 tokens with the highest probabilities. The generated text is limited to a maximum length of 100 tokens.
8. It decodes the generated output into text using the tokenizer.decode() method, which converts the token IDs back into text.
9. Cleaning up the text: The generated text may contain unwanted characters, punctuation, or extra spaces. You can use Python's string manipulation methods to clean up the text and make it more readable.
10. Finally, it prints the expanded text, which is the generated text based on the preprocessed input sentence.

Input and Output



Approach 2

My code defines a function called generate text that uses OpenAI's GPT-3 natural language processing model to generate text based on a given prompt. The function takes two arguments: prompt, which is a string representing the text prompt for the model, and model, which is the name of the GPT-3 model to use.

The function uses the openai.Completion.create method to generate text based on the prompt. This method sends the prompt to the specified GPT-3 model and returns a completion object. The max\_tokens parameter specifies the maximum number of tokens the completion should generate, while the **n** parameter specifies the number of completions to generate (in this case, only one).

The function then extracts the generated text from the completion object and returns it.

The code then defines an input text and a GPT-3 model name and uses these as arguments to call the generate text function. The output text is then printed to the console, which is the concatenation of the input text and the generated text.

Note that to use the OpenAI API, you must have an API key, which should be specified as a string in the openai.api\_key variable. The API key can be obtained from the OpenAI website. Also, the OpenAI API requires that the user have a valid account and have agreed to the OpenAI API Terms of Use.

Algorithm

1. Set the OpenAI API key in the openai.api\_key variable.
2. Define a function called generate\_text that takes two arguments:

prompt: a string representing the text prompt for the model.

model: a string representing the name of the GPT-3 model to use.

1. Inside the generate\_text function:
2. Use the openai.Completion.create method to generate text based on the prompt and GPT-3 model.
3. Specify the following parameters for the openai.Completion.create method:

engine: the name of the GPT-3 model to use.

prompt: the text prompt for the model.

max\_tokens: the maximum number of tokens the completion should generate.

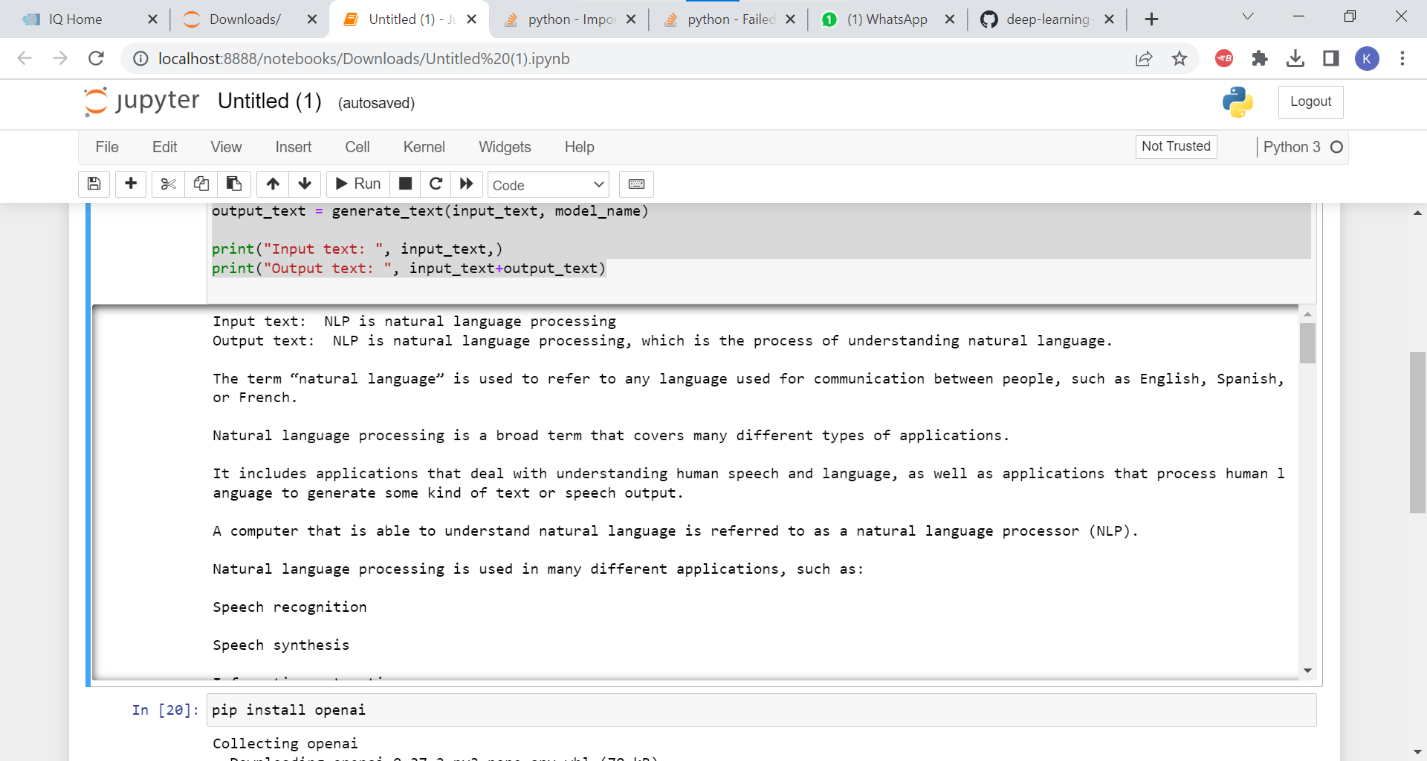
n: the number of completions to generate.

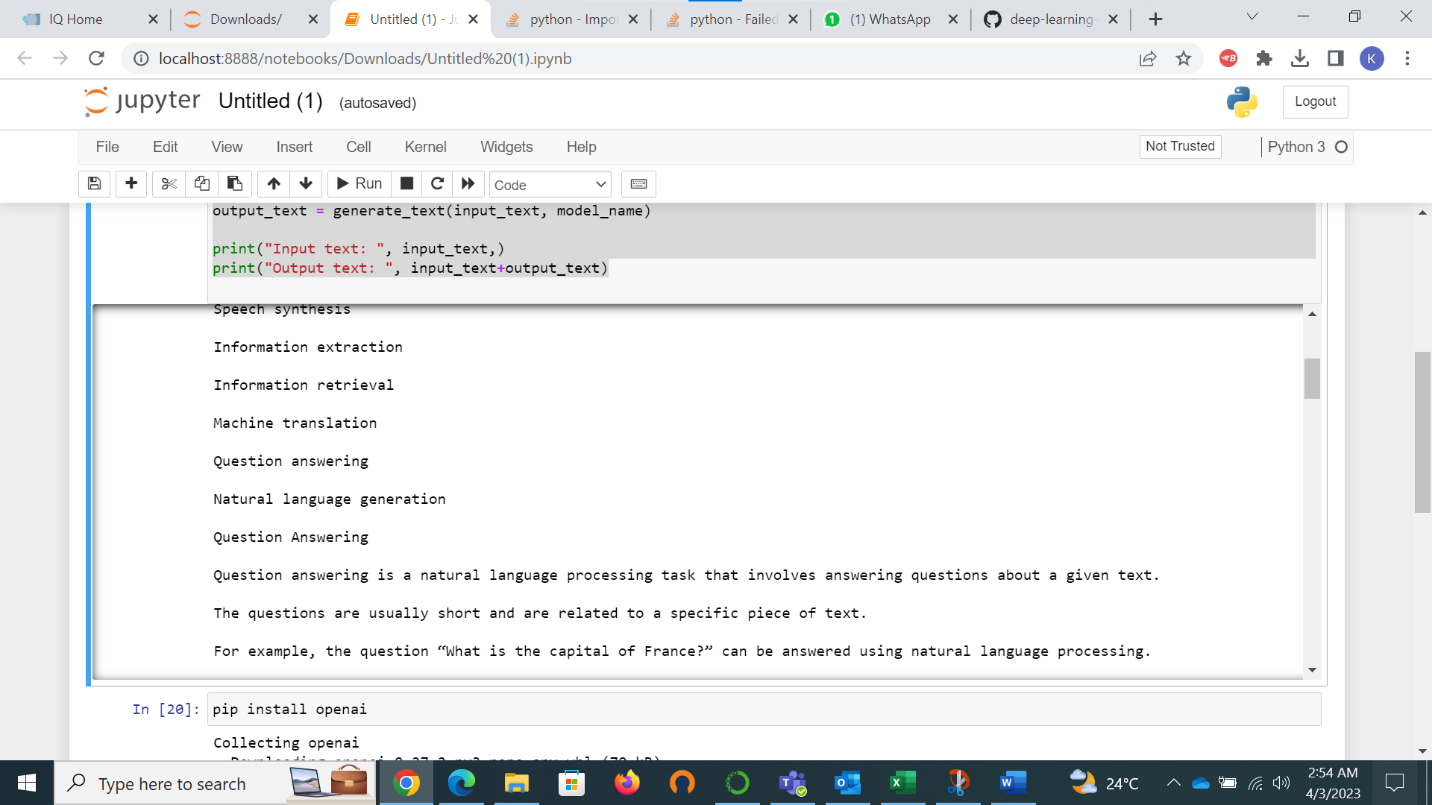
stop: a string or list of strings at which generation should stop.

temperature: a float representing the sampling temperature to use when generating completions.

1. Extract the generated text from the completion object and return it.
2. Define a string variable called input\_text with the value "NLP is natural language processing".
3. Define a string variable called model\_name with the value "davinci".
4. Call the generate\_text function with the arguments input\_text and model\_name to generate output text.
5. Print the input text and the output text to the console.
6. This algorithm describes the steps that the code takes to generate and print output text based on the given input text and GPT-3 model.

Snip of the input and output



Bottom of Form