**Open AI Introduction**

This module will cover different AI services.

|  |  |
| --- | --- |
| Module Introduction. | 2 |
| Module Requirements. | 2 |
| Before Coding. | 2 |
| AI Services Covered in this Module. | 3 |
| Project 1: The Translator App | 4 |
| Project 2: Image Generator App | 8 |
| Appendix A: The Assets | 14 |
| Appendix B: The Challenge | 15 |

**Module Introduction:**

Explore the realm of conversational AI with the ***OpenAI*** API and Python. These APIs open possibilities for dynamic human-like interactions. Tailored for all skill levels, this introductory module covers the basics of integrating the API, from making calls to handling responses, empowering you to add a layer of sophisticated interaction to your projects!

This module is meant to be accomplished within a single day in a GenCyber Camp time block.

**Module Requirements:**

Basic knowledge of the Python programming language. You will also need an editor such as IDLE which is currently bundled with Pythons installation. Make sure that the ***OpenAI*** library is also installed and breezypthongui.py file (included with this module) is in your project folder and ready to use before beginning. To install the ***openai*** library, using PIP, just run the following line:

**pip install –upgrade openai**

You will also need the ***playsound*** library. This will require a wheel upgrade, run the following:

**pip install --upgrade wheel**

**pip install playsound**

The final library this module will require is the SciPy library to write a wav file:

**pip install scipy**

**Before Coding:**

1. Create a folder named : ai***\_module (this will be the root of your project)***
2. Inside this folder create other folders named : **assets, images.**

Unzip the ***ai\_assets.zip*** file provided with this module into the assets folder and move the breezypythongui.py file into your projects root folder.

1. Create an account at [**https://openai.com**](https://openai.com) or receive a secret key from your instructor.

Note: ***OpenAI*** suggests setting a system environment variable. If you set up an environment variable for your secret key, you will have to modify this code only slightly.

AI Services Covered in this module:

GPT-3.5 Turbo:

GPT-3.5 Turbo is a highly advanced language model developed by OpenAI, representing the latest iteration in the Generative Pre-trained Transformer series. It excels in understanding and generating human-like text, capable of performing a wide range of natural language processing tasks. With 45 billion parameters, it surpasses its predecessor GPT-3 in terms of computational power, enabling more nuanced and contextually rich responses. GPT-3.5 Turbo is designed to facilitate various applications, from content creation to chatbots, offering developers a powerful tool for leveraging natural language understanding and generation in their projects. Its versatility and scale make it a cutting-edge solution for diverse language-related tasks in the field of artificial intelligence.

DALL-E:

DALL-E is an innovative artificial intelligence model created by OpenAI that specializes in generating images from textual descriptions. It belongs to the same family of models as GPT-3 but is tailored for visual content creation. With a dataset containing diverse images and corresponding textual prompts, DALL-E can generate novel and creative images based on textual input, demonstrating a remarkable ability to understand and translate abstract concepts into visual representations. This model showcases the potential of AI in the realm of creative arts and visual content generation, offering a unique and powerful tool for artists, designers, and other creative professionals.

**Project 1: Translator App (**translator.py**)**

**A screenshot of a computer

Description automatically generated**Using the ***BreezyPythonGUI*** library, we will be developing a quick form for use in taking input from a client and sending a request to the ***GPT3.5-Turbo API*** and receiving translated text back. The text will then be sent to a container on the form.

**Part A:**

Before we interact with the AI, we need to build out the form that we will be using to send and receive the data.

Create the translator.py file. This will be the only file required you will be writing for this application.

**Imports:** Add the following imports to the top of your file after any documentation

A close up of a word

Description automatically generated

A close-up of a computer code

Description automatically generatedNext Declare the class header making sure to inherit from EasyFrame and not the base Object class. Inside the class, declare the constructor method. As this is our main and only class, it should only receive self as a parameter.

The Next three lines cover the following.

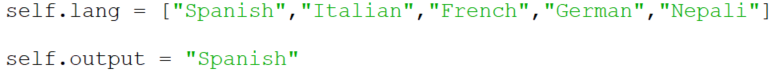
1. Call the ***EasyFrame*** constructor, set the title to “Language Translator” and set the width and height of the window frame to 300 and 380 respectively.

2***. self.api\_key*** is where you would store your ***OpenAI’s*** Secret Key as a string. If you set your secret key as an environmental variable, you can omit this line.

3. Set ***self.client*** to an instantiation of the ***OpenAI*** Object. Again, omit the key if you used the environmental variable.

A close-up of a computer screen

Description automatically generated



***self.lang*** is where we set the available languages that our application covers. While ***ChatGPT*** does not know it can translate now, it knows plenty of languages, such as: English, Spanish, French, German, Chinese (Simplified and Traditional), Japanese, Korean, Russian, Italian, and Portuguese.

***self.output*** will hold the language the text will be translated to. As default, it is set to Spanish, but could also be an empty string.

A close-up of a computer code

Description automatically generated

The ***addLabel*** methods will place a label in the form at the specific locations. The ***inputField*** method creates an input field. We store it in the ***self.inputField*** property so we can access its value later when we need to run our ***translate*** method. ***addListbox*** adds a widget to the window that allows a user to make a choice between several options.

A close-up of a computer code

Description automatically generated

To set the available options for the ***listbox***, we use a for loop that runs for each of the items in our ***self.lang*** list property. The ***insert*** method requires the ***index*** position and then the ***value*** that you would like to display as parameters.

With that set, it is time to create an area for the translated text to be displayed. First create the Label to show that this is the translation section.

A close-up of text

Description automatically generated

Then using the ***addTextArea*** method, you will set the area which will display the translated text to the user. This will be saved in the ***self.outputText*** property so we can access it later. Note we will not be accessing the label for any reason, so we do not need to save it as a property.

At this point the window display should be completed in the constructor for this class.

A close-up of a computer code

Description automatically generated

Create the main function as shown above to run the mainloop of the class. To test what you have done up to this point, you need to locate the line of code below and comment this section of the code out. Make sure to provide the closing ) for the original code since the original ) is also commented out.



Then execute the code and make sure you get everything showing up correctly. It should look like the image below:

A screenshot of a computer

Description automatically generatedOnce you have tested and debugged your code up to this point, you can uncomment out the line we commented out again. Make sure to remove the ) that we added.

The reason this code had to be commented out is we have not yet created the method for this section. Let’s do that now.

A close up of a text

Description automatically generated

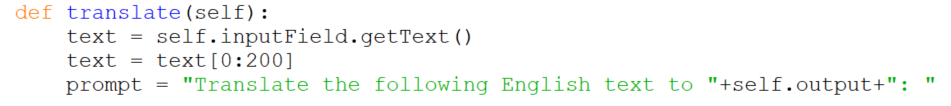
Create the above class method. This will set the output property to the chosen language in the list box every time a user clicks a new language.

**Part B:**

To finish this application, we have only two more steps to get through. Revisiting the constructor method, create a button that will run a translator method which will send out a call to the AI and receive a response.



While we do not need to re-access this button in our current design, if you were to modify this application to limit the number of requests any one person can send, it is always a good idea to save buttons as a localized property that can be manipulated later. For that reason, we sill save it to the ***self.execute*** property. Set the text to Translate and then the command to the ***self.translate*** method we will work on next.



Declare the translate method like any other normal method. It will require no parameters other than self as everything it needs to run is available as properties of the class itself.

The next two lines, gets the text that was entered by the user form the ***inputField*** and then limits it to 200 characters. In normal operation you may not want to limit a commercial application like this but due to costs, we need to make sure we keep the overhead for this educational assignment as low as possible. Limiting to 200 will do just that, as every word is considered a token as well as each specialized symbol.

Then create the prompt that we will be sending to the GPT AI. We add the “Translate the following English text to” then provide the language we want it to translate the text to. This will tell the AI exactly what we want it to do. The next chunk of code will be the response object that will be sent to the processor.

A computer screen shot of a computer code

Description automatically generated

As soon as this create method is called, it will send out a request to the AI so make sure you have all your code ready to test before you start running your application.

The model we will be using is the GPT 3.5 Turbo. Currently 4.0 is out but in testing phases and the cost per token is a decent jump up in price so we will be limiting ourselves to the 3.5 version.

The messages list contains a list of dictionaries. It requires positions for both the system (AI) and the user (client). The prompt goes in the systems content value and then the text from the ***inputText*** element goes in the content value of the user dictionary.

We can also set the number of max tokens that we want the service to limit itself to. This also helps keep the overall overhead cost down on the application.

The last piece of this puzzle is to set the text on the text area of our application with the following line of code:



That’s it for this program! Feel free to debug and test it with any phrases you would like. Text generation is by far the cheapest of the AI services that we will be demonstrating in this module.

**Project 2: Image Generator App (**imagegenerator.py**)**

A computer screen shot of a question mark

Description automatically generatedUsing the ***BreezyPythonGUI*** library, we will be developing a similar form as the last project for use in taking the image idea input from a client as well as selecting the style of image that should be developed and then sending a request to the ***DALL-E API*** to receive our generated image back. The image will then be stored in the ***images*** folder and then loaded into the image label on the form for display.

**Part A:**

Before we interact with the AI, we need to build out the form that we will be using to send and receive the data.

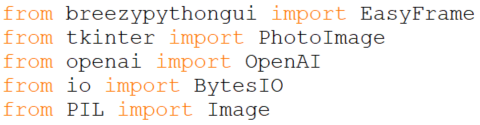
Create the imagegenerator.py file. This will be the only file required that you will be writing for this application.

**Imports:** Add the following imports to the top of your file after any documentation

A group of black and orange text

Description automatically generated

The base64 import is to be able to convert the image data we will receive back. The os import is to access key system methods. Both the random and String imports are required to create a random filename for use in saving the images that are generated.



The second set of imports we will be adding to this program call individual classes that we will need to import. The first three you should be familiar with after finishing Project 1. BytesIO and PIL’s Image class will be used to save the image data to a file in the images folder.

A computer code with text

Description automatically generated with medium confidenceNext Declare the class header making sure to inherit from ***EasyFrame*** and not the base Object class. Inside the class, declare the constructor method. As this is our main and only class, it should only receive self as a parameter. Call the ***EasyFrame*** constructor, set the title to “Image Generator”. Since we added the dimensions n the ***EasyFrame*** constructor last time, this method will show how to add the dimensions of the width and height in after the fact, using the ***setSize*** method. Set the width and height of the window frame using this method to 300 and 540 respectively.

The following set of properties must be added to the constructor method.

A white background with green text

Description automatically generated

As with the last application, you will enter to key in the appropriate properties values and then create the ***OpenAI*** object stored in ***self.client***. The 3 properties after are for use in the request that we will send to the ***DALL-E*** service. The first, ***self.imageSize*** sets the size of the Image. For ***DALL-E*** 2, you can use larger images but for this course please limit it to the ***256x256*** size as this should keep the cost of each image generated to just under **2 cents**. The model is the model we are requesting, on your own you can discover the other available models you can use when working on your future projects that require such a service. For this module though do not use a different value. Finally add your username in the ***self.user*** property. This is required by ***DALL-E*** to help cut down on the improper usage of the service.

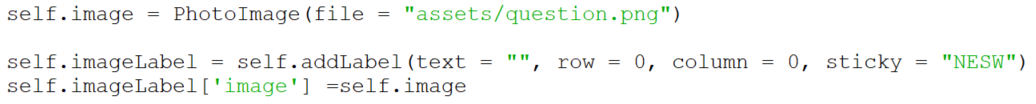
Like how we setup the languages in the last application, create a list called ***self.styles*** which should hold some of the available styles that ***DALL-E*** can work with. Note: These are only a handful of styles that the service can handle.

A close up of text

Description automatically generated

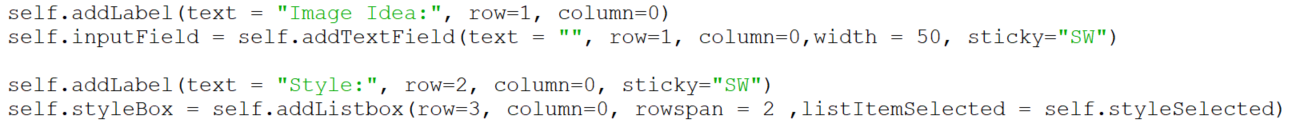
Make sure to create a ***self.style*** property that is set to the first style in the list as default. This value will be updated when the user clicks a different option in the list box.

Using ***tkinter***’s ***PhotoImage*** class, load the question mark image found in the assets folder into your application and store it in the ***self.image*** property. This property will be updated on every successful image generation and will show the latest one pulled.



Then create the label that we will be using to display the question mark and other generated images. Finally add the image just loaded to the image label.

Like the translator app, the following 4 lines have been slightly modified but behave in the same manor. A label and text field are added to the app but this time instead of holding the string to translate, it will hold the image idea to pass to the service. Then another label and another list box. This time though the box is saved in self.styleBox and the method to run when a user clicks an item is self.styleSelected.



The next code block covers the loop that will add each of the styles from our list to the list box. After the loop is done running, make sure to set the initial selected index of the list box to 0.

A close up of words

Description automatically generated

Now, like in the translator app, we will add a quick method that is triggered by a click on the list box. Its entire function is just to set the ***self.style*** property equal to the string that corresponds with the same index in the list box.

A math equation with blue text

Description automatically generated with medium confidence

Finally write the main method for the application, calling the classes mainloop and then make sure to call the main function at the bottom of the file.

A close-up of a computer code

Description automatically generated

A screenshot of a question mark

Description automatically generatedAt this point you should be able to run the application. It should look like this image. If it doesn’t run or doesn’t look like this, go back over the code until you get it to function and resemble the image.

Once you have reached this point, it is time to build the interactivity for the AI service.

**Part B:**

To begin this section, add the button to the constructor function. It should be calling the method ***self.fetchImage*** and should contain the text “Generate Image”.



That was the easy section of this part (b), the rest of this part will be dedicated to creating the ***self.fetchImage*** method. Unlike the last application though, this code block is much more involved. To start, lets declare the method ***fetchImage*** in our class.



A screenshot of a computer code

Description automatically generatedLike the translator application, get the text from the input field and save it in a temporary variable called ***text***. This will be the subject for our prompt.

Then Declare the ***prompt***, which we will store as a tuple with a single string value inside. The String has the format “Subject: stuff Style: stuff” with a single space between the values. As this is one string, we can put it on 2 separate lines without having to use a concatenation operator. Add in the ***text*** variable and the ***self.style*** property and we have a completed prompt! Must note that when using ***DALL-E*** 2 the prompt is limited to 1000 characters, and with ***DALL-E*** 3, 4000 characters.

The next dictionary we must make contains the image parameters. Fill in the dictionary using the properties we created when we were building out the constructor class.

Using the ***random*** and ***string*** libraries, the above line will generate a random string of lowercase ascii characters that is 10 characters in length.

A computer screen shot of a program

Description automatically generated

The update method sets the preferred response format from the service to base64 as a JSON object (JavaScript Object Notation). The try and except blocks will attempt to generate the image by sending the request to the AI service, and if not, will print the generated error.

Next create a list that will contain any images returned by the service as it will return a list, even for a single image being returned. Then loop through the items in the response received from the AI service. Append this to the list.

A close-up of a computer screen

Description automatically generated

Then create an empty list to contain the generate image objects.

If items are inside the image\_data\_list list, then loop through them. The following code will loop through the items in this list. As we are only expecting a single image returned, we have only generated a single filename. If you request multiple images at once, you would just change the functionality to generate a random string for every image returned.

Append the Image object to the image\_objects list, save the filename into the images folder. All print statements are just for debugging purposes.

A computer code with text

Description automatically generated

If there is no data in the ***image\_data\_list***, print to the shell that no image data was obtained. Again, for debugging purposes.

A white background with green text

Description automatically generated

Outside of the if statement, to wrap things up, once again load the image using ***tkinter***’s ***PhotoImage*** class and save it to the ***self.image*** property.

Then use that property to set the image of the ***imageLabel***.

That’s it for this program! Feel free to debug and test it with any image request combinations you would like. Be mindful though that every image generation will cost the API KEY holder around $0.016 which between debugging and running a few tests, can add up quickly. This is substantially more expensive than the standard text generator.

A screenshot of a drawing

Description automatically generated A screenshot of a computer screen

Description automatically generated A screenshot of a computer screen

Description automatically generated

Appendix A: The Assets

All images with an outside link are licensed under [Creative Commons by Attribution 4.0](https://creativecommons.org/licenses/by/4.0/), unless otherwise stated.

question.png:

Image generated by DALL-E.

Breezypythongui.py:

Library built off the TKinter library, created by Ken Lambert

<https://lambertk.academic.wlu.edu/breezypythongui/tutorial-for-breezypythongui/>

Appendix B: The Challenge

**Project:**

Using the lessons learned in this module, you will be blending the concepts together creating a therapist style application.

1. The user should be able to input a sentence to talk to an AI Therapist. This sentence should be proceeded with the string : “Pretend to be a therapist: “.
2. With the text in the input field, the client should then be able to hit a button that will send out the data, much like Project 1, to the service and get the Therapist’s response back.
3. Display this response.
   1. Sent to the text area for output.
4. Finally generate an image based upon the initial prompt sent by the user. Use a random style for the image.
5. Display this image in the program.