# Create a GUI to extract Lyrics from song Using Python - ProjectGurukul

#### **About Lyrics extractor:**

A karaoke or a sing along is a great way to have fun, isn't it? We see the lyrics on a screen and the music plays on the background with no voice and we sing to the tune seeing the lyrics. Similarly, why don’t we try implementing one such project to extract the lyrics?

#### **Python Lyrics Extractor:**

Here, we create a simple lyric extractor and add a GUI to it.

#### **Project Prerequisites:**

The project is very simple to work on and uses the modules: tkinter, google api client, beautifulsoup and requests. We use google api client to make a connection and extract the desired search result and we create the GUI using tkinter. Requests and beautiful soup are used to scrape the retrieved result to obtain the lyrics.

Requests and Beautifulsoup are built-in libraries.

Check if tkinter is available in python, since it is a built-in library.

import tkinter

If the library exists, it will import without an error. In case an error appears, install tkinter using the command: sudo apt-get install python3-tkinter for linux users and windows users can reinstall python. Alternatively windows users can follow the steps given in [Tkinter installation on windows](https://tkdocs.com/tutorial/install.html#install-win-python).

Install the google api client library using pip: pip install google-api-python-client

In case an error stating test resources is not available, install it using sudo apt install python3-testresources on linux systems.

#### **Download Lyrics Extractor code:**

You can download the source code for the Lyrics Extractor in the given link: <add link>

#### **Project File Structure:**

Let us look at the flow of the project

1. Importing tkinter, requests, beautifulsoup and googleapiclient modules
2. Obtaining the key and creating a programmable search engine
3. Defining the lyrics extractor function.
4. Create the GUI.
5. Activating the function using a button.

Now for the implementation in detail.

Feel free to play with the values and change your input methods.

##### **Importing tkinter, requests, beautifulsoup and googleapiclient modules:**

#ProjectGurukul's Guide for lyrics extractor

from tkinter import \*

from tkinter import messagebox

from tkinter import scrolledtext

from googleapiclient.discovery import build

import requests

from bs4 import BeautifulSoup

**Code Explanation:**

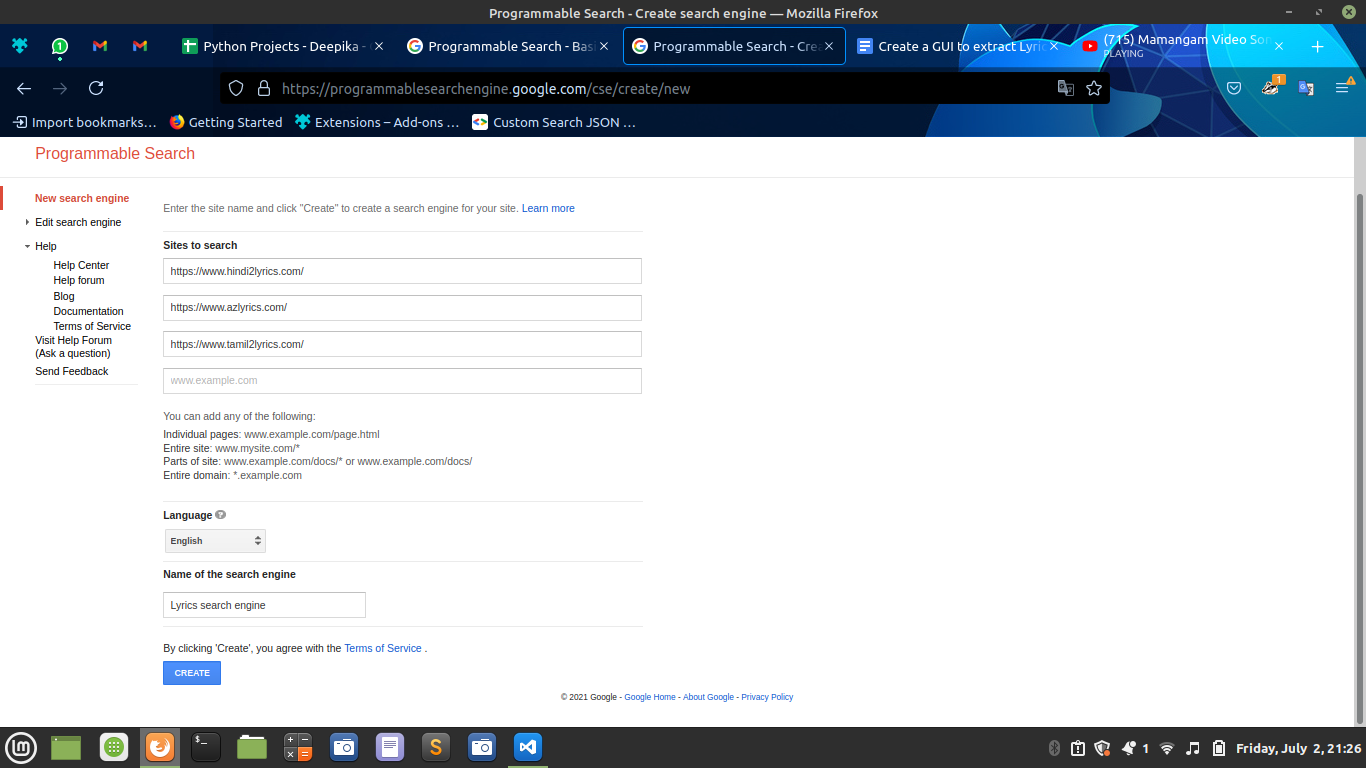
* **from tkinter import \*:** Tkinter contains the GUI widgets to create the user interface
* **from tkinter import messagebox:** To create prompts to alert the user, we use messagebox. It contains prompt to ask the user for yes or no questions, show a warning and errors etc
* **from tkinter import scrolledtext:** The lyrics can be longer than the set dimensions of the window of the application. Hence we create a scrolling text widget to allow scrolling of the text widget and display the lyrics
* **from googleapiclient.discovery import build:** To establish a connection with the search engine we will create later.
* **import requests:** To open the website and retrieve the webpage
* **from bs4 import BeautifulSoup:** To scrape the website to obtain necessary content from the website

##### **Obtaining the key and creating a programmable search engine:**

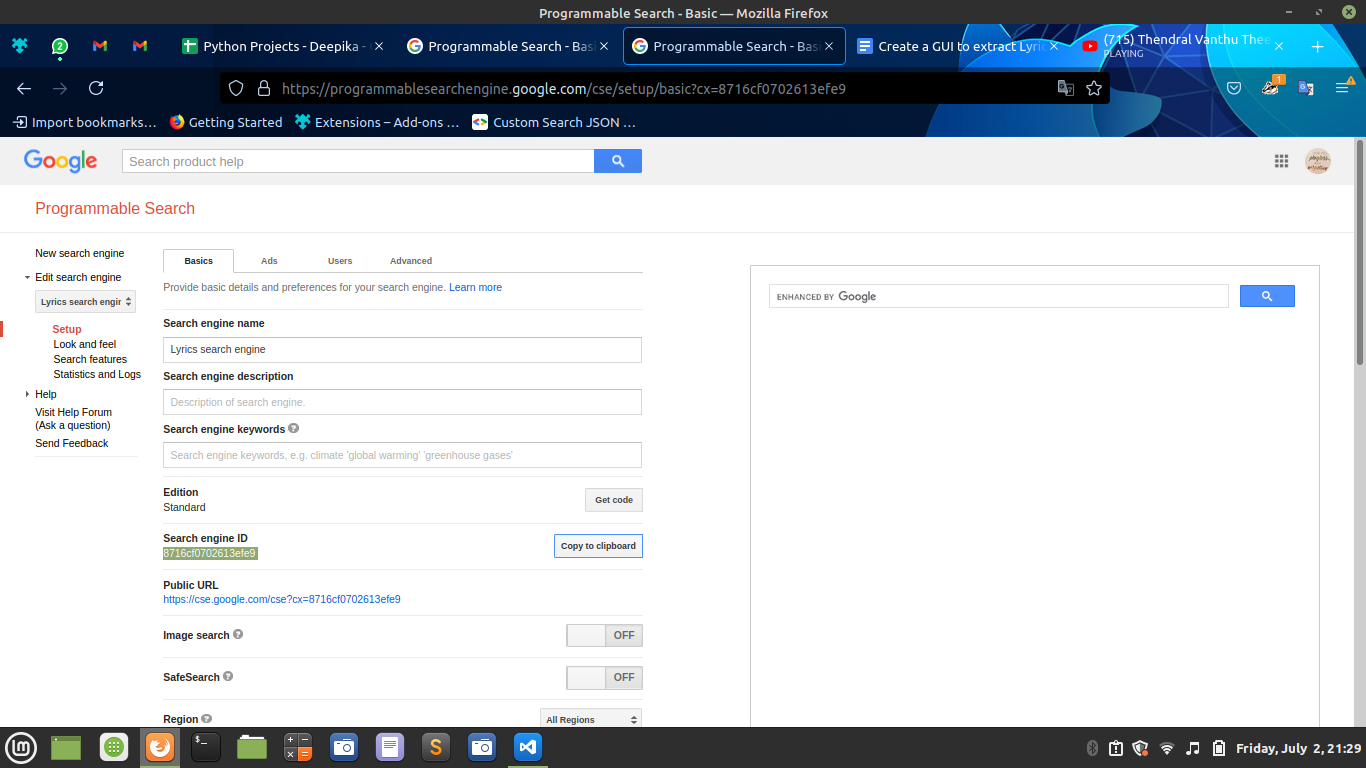
To create the lyrics extractor, we need to do 2 steps. The first step involves creation of a programmable search engine and the second step is obtaining the API key.

1. **Creation of a programmable search engine:**

To create a programmable search engine, head to [Programmable search engine using control panel](https://programmablesearchengine.google.com/cse/all).

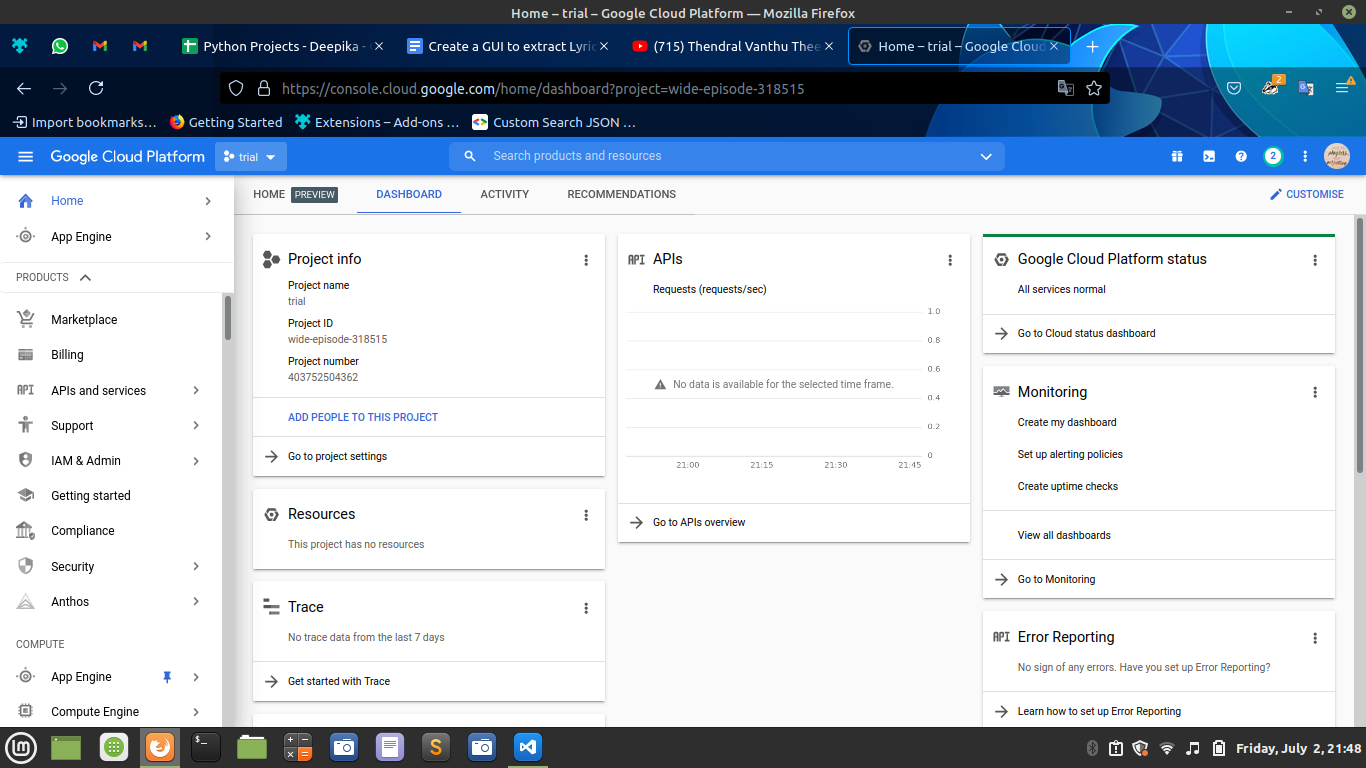


Click on **‘New search engine’**  to create a new engine. In the next page, under sites to search add the links of the lyrics websites of your preference and give the search engine a name. Hit on **create.** Next we need to find the ID of the search engine. To do this, select **control panel** button and copy the search engine ID

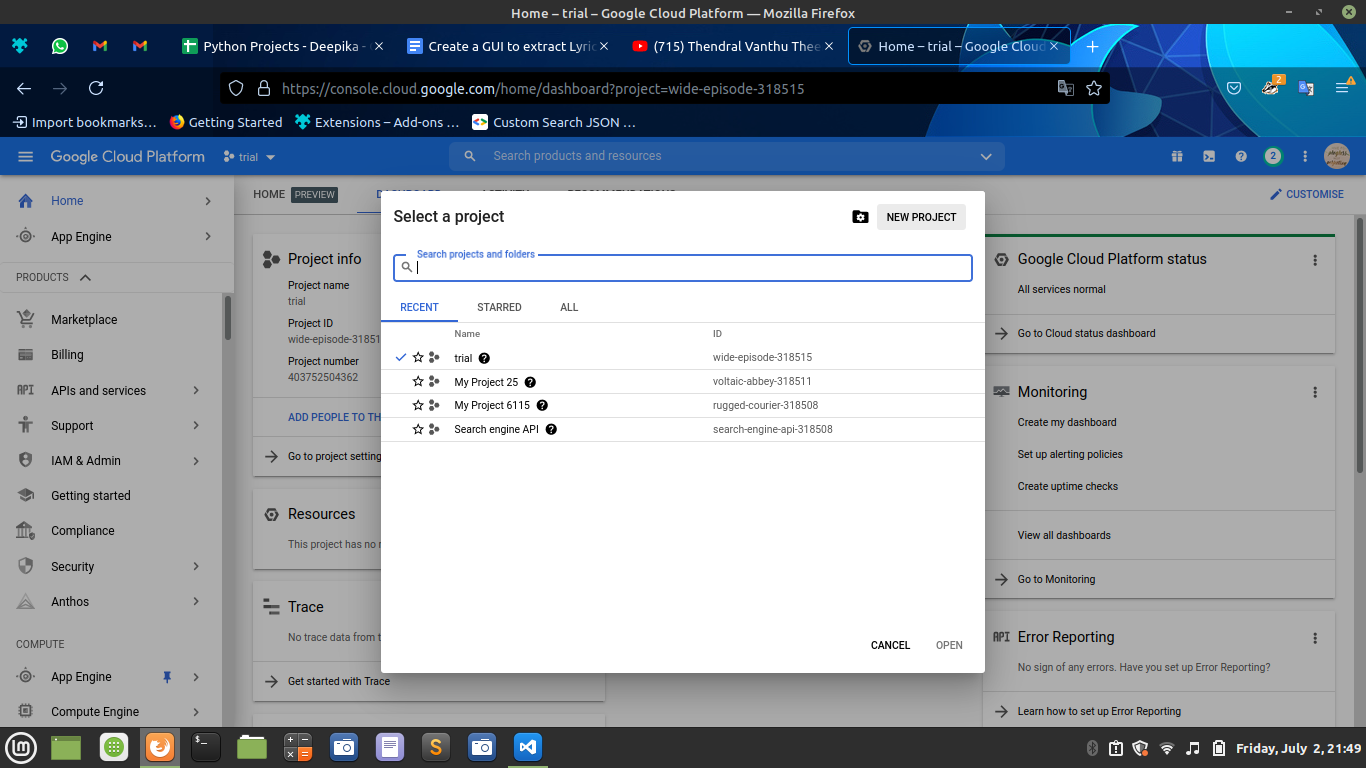


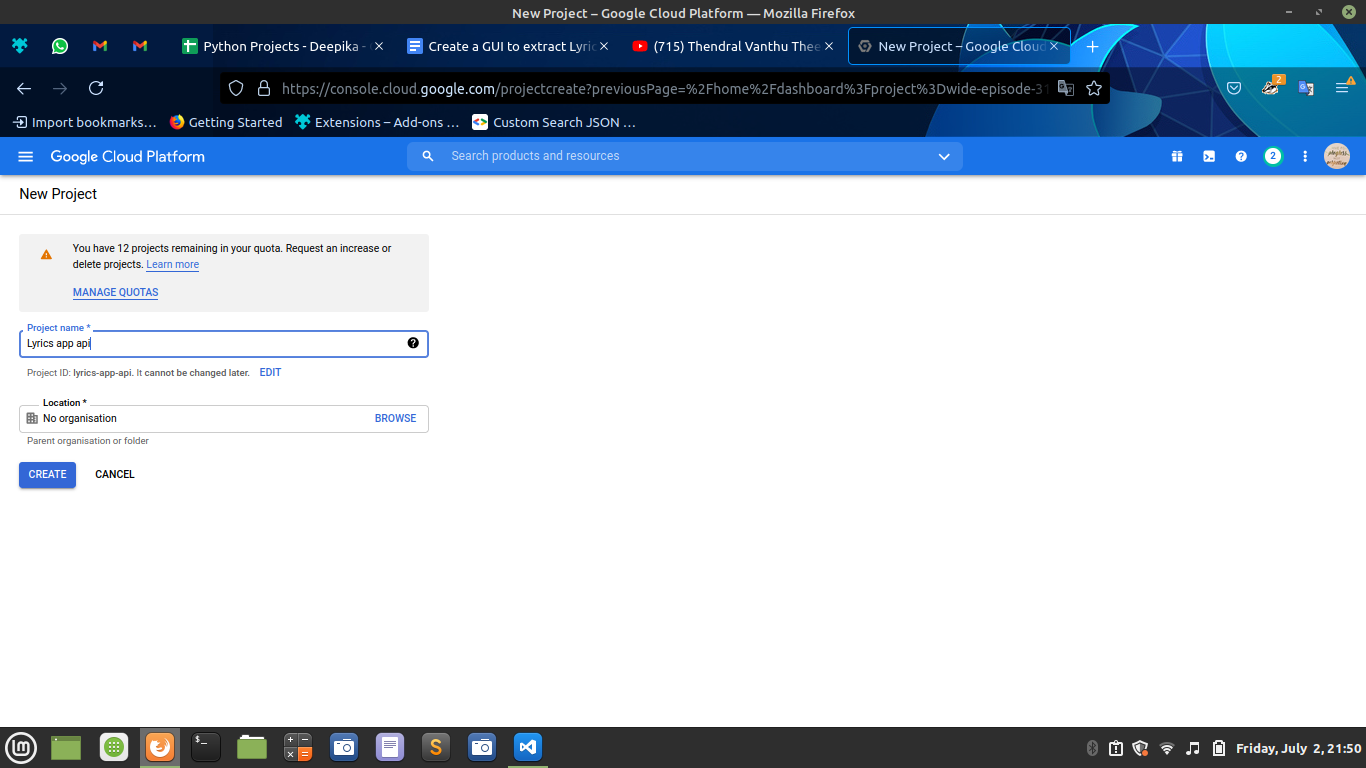
1. **Obtaining the API key:**

After obtaining the search engine ID, head to [Google developer console](https://console.cloud.google.com/). The following window will appear.



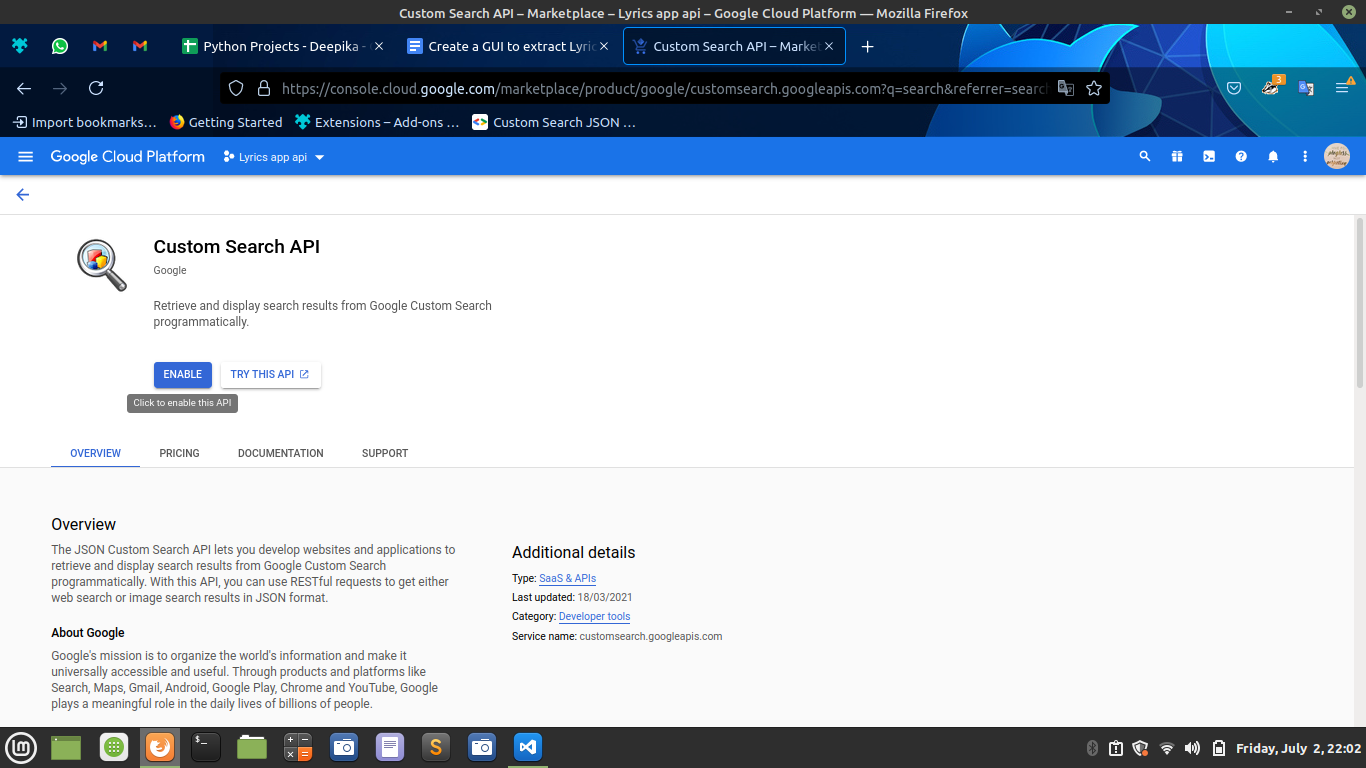
Select **New Project** from the drop down list and name the project



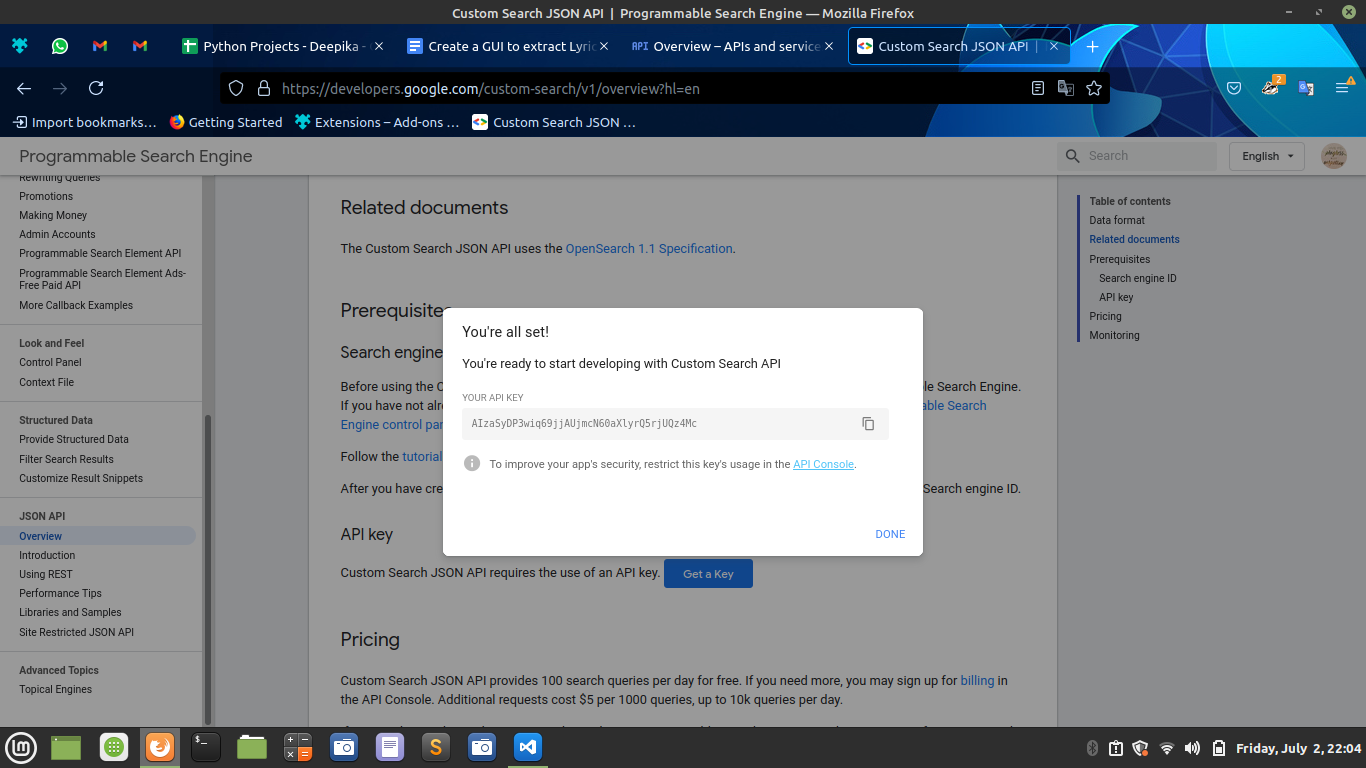


Your project will be created in a few minutes. To work in the created project, select the project from the dropdown list next to Google Cloud Platform or hit ‘Select project’ in the notifications.

To create a search API, type **Custom Search API**  in the search bar. When the following window appears, click on enable.



You have successfully created the API. To get the key, go to [Custom JSON API](https://developers.google.com/custom-search/v1/overview?hl=en). Scroll down until you see ‘Get a Key’. Select the option and choose the API you previously created.



Copy the API key. Now google’s custom search engine API provides us with a limit of 100 queries a day. So if you exhaust the limit and cannot wait a day, create another project and enable the API.

##### **Defining the lyrics extractor function:**

Before declaring the function, declare and initialise two variables,

#Create the extractor function

def lyrics\_extractor():

#Read the name of the song

song = song\_entry.get()

#Raise a warning if no input if given

if len(song) <=0:

messagebox.showerror(message = "Enter a song name" )

return

else:

#connect the API with the search engine

lyric\_object = build("customsearch", 'v1', developerKey=api\_key).cse()

#execute the search

result = lyric\_object.list(q=song, cx=cse\_key).execute()

#Obtain the first result of the 10 retrieved

try:

first\_value = next(iter(result["items"]))

except:

messagebox.showwarning(message = "Lyrics for "+song+" were not found" )

return

#Send a request link to extract contents of the first website

result = requests.get(first\_value["link"])

#To obtain individual components in the html format

soup\_lyrics = BeautifulSoup(result.content, "html.parser")

#Find the language using the link

if ("tamil" in first\_value["link"]) | ("hindi" in first\_value["link"]):

#if tamil, read the lyric from the div tag containing print-lyrics class

lyrics = soup\_lyrics.find(class\_ = "print-lyrics")

elif "azlyrics" in first\_value["link"]:

#if english, read the lyric from the div tag containing no class and id

lyrics = soup\_lyrics.find("div", {'class':'', 'id':''})

#Display the output

display\_lyrics = scrolledtext.ScrolledText(lyrics\_app, width = 60, height=10, font=("Ubuntu Mono",10), bd=0, bg="seagreen3")

display\_lyrics.insert("1.0",lyrics.text)

display\_lyrics.config(state=DISABLED)

display\_lyrics.grid(row=2, column=0, columnspan=3)

**Code explanation:**

* **def lyrics\_extractor():** Declaration of the lyrics\_extractor function.
* **song = song\_entry.get():** Read the contents of the entry widget
* **if len(song) <=0:** If the user fails to provide an input, an error message is generated using the showerror() messagebox prompt which contains a parameter, message, containing the text to be displayed.
* **lyric\_object = build("customsearch", 'v1', developerKey=api\_key).cse():**  To create a connection between the custom search engine created, we use build. build() has three parameters: customsearch - service of the app, v1- version of the app, developerKey- API key created. cse() invokes the connection and creates the object assigned to lyric\_object.
* **result = lyric\_object.list(q=song, cx=cse\_key).execute():** list is a method to retrieve the requested search results. It contains a query - the song and the cx, which is the search engine id. execute() initiates the search and retrieves 10 results by default.
* **try...except:** The requested song lyrics may or may not be found. Hence we check if the result variable contains actual search results. In case none is obtained, display a prompt using showwarning() messagebox. We obtain the first search result by iterating through the items in the JSON and using next() we obtain the first result
* **result = requests.get(first\_value["link"]):** The first result is most likely the best result and relevant result. To open the website in first\_value, we extract the link and use requests.get() to open the link.
* **soup\_lyrics = BeautifulSoup(result.content, "html.parser"):** The result obtained will contain the HTML format of the website with the content. Thus to be able to extract the necessary content, lyrics, we use BeautifulSoup() to parse or read through the html content. It contains two parameters, result.content which extracts the html content and “html.parser” indicates the way of parsing data.
* **Test condition:** Since the search engine created in the tutorial allows to retrieve lyrics for tamil, hindi and english songs. Hence there are 3 websites in use, two of which have the same webpage layout (tamil2lyrics and hindi2lyrics). Since the web page layout changes between different websites, to extract lyrics, we need to analyse the HTML content to find the class where the data is present. For tamil2lyrics and hindi2lyrics, the lyrics are present in the div tag of the class = “print-lyrics” and for the english songs in azlyrics the lyrics are in the div tag with no class name and ID. find() method extracts the content from the given tag or class name.
* **display\_lyrics:** To display the extracted lyrics, we need a textbox. But a textbox may not be sufficient since the lyrics may be longer than the screen length, we cannot be certain of dynamically resizing the window. Hence we use a scrolledtext. ScrolledText() text widget to contain the lyrics in a scrolling text widget. Thus, the window size need not be altered. To display the lyrics, we extract only the text without HTML tags using lyrics.text and insert it using the insert command and disable the text widget to make it read-only. A widget is only visible when it is positioned. Thus we position it using grid, which divides the entire screen space into rows and columns. So we position the widget in row 2 and column 0 of the screen. Columnspan allows a widget to span or be visible across multiple rows without altering the layout for other widgets. Thus a columnspan of 3 allows the widget to be set across three columns.

##### **Create the GUI:**

#Create the user interface for the application

lyrics\_app = Tk()

lyrics\_app.geometry("500x300")

lyrics\_app.title("ProjectGurukul's Lyrics extractor")

lyrics\_app.config(bg="seagreen1")

#Specify the entry fields

song\_label = Label(lyrics\_app, text="Enter song name: ",bg="seagreen1")

song\_label.grid(row=0, column=0, padx=20, pady=20)

song\_entry = Entry(lyrics\_app, width=40)

song\_entry.grid(row=0, column=1)

**Code explanation:**

* **lyrics\_app = Tk():** Create an object for the application
* **lyrics\_app.geometry("500x300"):** Define the dimensions of the application window
* **lyrics\_app.title("ProjectGurukul's Lyrics extractor"):** Assign a title to the application
* **lyrics\_app.config(bg="seagreen1"):** Give the application a background colour using ‘bg’ and configure. Default background colour is gray85
* **song\_label = Label(lyrics\_app, text="Enter song name: ",bg="seagreen1"):** Define a label to provide non editable text incapable of being copied. Lyrics\_app is the window of the application, text is the text to display and bg is the background colour of the label.
* **song\_label.grid(row=0, column=0, padx=20, pady=20):** Position the label in grid format in the row 0 and column 0, padx and pady defines the distance from the left and top margin respectively
* **song\_entry = Entry(lyrics\_app, width=40):** Read user input through entry widget. Mention the width of the widget using width parameter and specify the window of the application where it will be placed on
* **song\_entry.grid(row=0, column=1):** Place the widget in column 1 of the same row

##### **Activating the function using a button:**

#Button for lyrics

get\_lyrics\_button = Button(lyrics\_app, text="Get Lyrics", command=lyrics\_extractor)

get\_lyrics\_button.grid(row=1, columnspan=2,pady=20)

#Terminate the app upon closing

lyrics\_app.mainloop()

**Code explanation:**

This snippet of code is optional

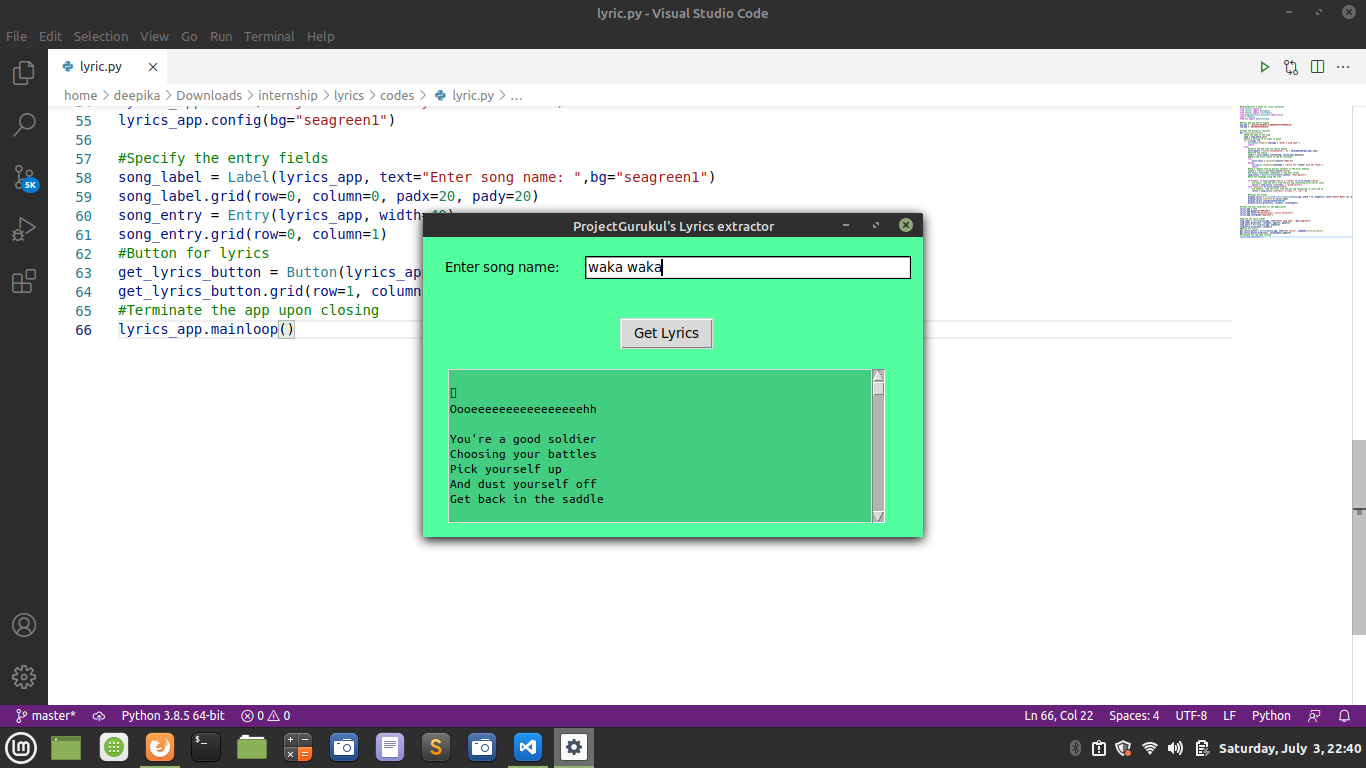
* **get\_lyrics\_button:** To use the lyrics\_extractor function, we use a button to activate the function when the user selects it. Hence we define a Button widget with the parameters, lyrics\_app: window or screen of the app, text: name of the button and command: calls the function
* **get\_lyrics\_button.grid(row=1, columnspan=2,pady=20):** Center the button using columnspan to position it across two columns in row 1, with a padding of 20 from the top margin
* **lyrics\_app. mainloop():** To activate and display the app we invoke this command. When the user selects quit, the control flows past this command, thus destroying the app

##### **Extras:**

To find the div tag containing the lyrics, first print the contents contained in the result of using BeautifulSoup. After printing the result on the console, scroll along to the section containing the lyrics and check if the div tag contains a class name or an ID. If yes, then use the same to extract the lyrics.

#### **Project output:**

Enter the inputs and view the output:



#### **Summary**

Thus we created a simple Lyrics Extractor from scratch. The project is a great introduction to webscraping, GUI designing using tkinter and usage of google api client to create a custom search engine.