VOICE COMMAND WEB SCRAPER

CASRON JOHNSON (301041836)

CHRISTO JACOB (301108924)

SHANIQUO MCKENZIE (301024574)

MUSTAFA BUTT (822392403)

MICHELLE FAJARDO (301097601)

APRIL 6, 2021

GROUP #1

Table of Contents

Abstract

The purpose of the voice command app is to scrape weather data from a dynamic website, taking country & city parameters, & returning forecasts as strings. Taking of parameters and returning of forecasts will be done with speech recognition. The required software and hardware are easily available and easy to work with.

Introduction

This project implements voice command and speech recognition using the speech_recoginition and pyttsx3 modules. Beautifulsoup4 module is also used to scrape the web and selenium to run the web driver to grab data from a dynamic website. The application will communicate with the user, similar to Apple Siri. Pyttsx3 will convert strings into voice whenever we want to output data to the user. When the user inputs his/her command using their voice, the app will call a function to scrape weather data from the web using the country and city input from the user. To avoid any browser from opening, we will use an option variable to suppress the browser's action. We will use a variable to represent the Chrome browser. When the application is run, the user's microphone will go into listening mode. Here, the user will be presented with voice instructions to speak the country and city's name. The app will raise any errors if a country or city is not validated.

Body of report

Fig 1: *MacOs Compatibility. Can operate on both Windows and Apple*

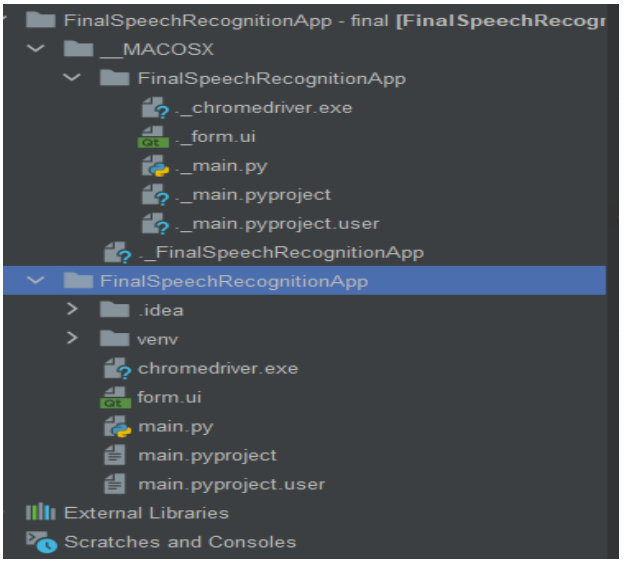


Fig 2: *Import statements include BeautifulSoup, speech_recognition and more libraries*

```
import speech_recognition
import pyttsx3
from selenium import webdriver
from bs4 import BeautifulSoup
import pycountry
import requests
import random
import sys
from PyQt5.QtWidgets import QApplication
from PyQt5 import uic
import os

path = os.path.join(os.path.dirname(__file__))

MainWindowUI, MainWindowBase = uic.loadUiType(
    os.path.join(path, 'form.ui'))

```

Fig 3: *SpeechWebScraper class inherits from 2 GUI Classes. The constructor initializes & loads the required components. ROBOT_VOICE method allows for text-to-speech speaking*

```
class SpeechWebScraper(MainWindowBase, MainWindowUI):

    def __init__(self, parent=None):
        MainWindowBase.__init__(self, parent)

        self.setupUi(self)
        # Create a variable that will represent as robot app to communicate with the user, just like Siri
        self.robot = speech_recognition.Recognizer()
        # Create a variable for speaking using pyttsx module
        self.audio = pyttsx3.init()
        # Create a self.robot voice to communicate with the user
        self.voices = self.audio.getProperty('voices')
        self.audio.setProperty('voices', self.voices[0].id)
        self.pushButton_start.clicked.connect(self.menu_selection)
        self.pushButton_stop.clicked.connect(self.stop_event)
        self.textBrowser_conversation.append("""
                ++++++++++▄▀▀▀▀▀▀▀▀▄++++++++++
                +++++++++▄▀▀▀▀▀▀▀▀▀▀▄+++++++++
                +++++++++▀▀▀▀▀▀▀▀▀▀▀▀+++++++++
                ++++++++▄▀▀▀▀▀▀▀▀▀▀▀▀▄++++++++
                ++++++++▀▀▀▀▀▀▀▀▀▀▀▀▀▀++++++++
                +++++▄▀▀▀▀▀▄▀▀▀▀▀▀▄▀▀▀▀▀▄+++++
                +++++▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀+++++
                +++++▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀+++++
                +++++▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀+++++
                +++++▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀+++++
        """)
        self.continueLoop = True

    # Create a function to read the words and say it out loud
    def ROBOT_VOICE(self, words):
        self.audio.say(words)
        self.audio.runAndWait()
```

Fig 3: *Weather_scraper method dynamically scrapes the given website using Selenium and BeautifulSoup. The method takes 2 string parameters, city & country, for the URL. Selenium loads the full pages code and BeautifulSoup parses the webpage's code to extract the information we need.*

```
    # Create function to scrape weather data from the web, takes country & city parameters, & returns forecast as string
    def weather_scraper(self, country: str, city: str):
        # Variable holds path to webdriver for selenium use
        PATH = "./chromedriver.exe"
        # Create variable to store url with country and city appended which will be opened
        URL = 'https://www.timeanddate.com/weather/' + country + '/' + city
        # Option variable to pass driver object to suppress the opening of the browser
        option = webdriver.ChromeOptions()
        option.add_argument('headless')
        # Variable represents Chrome browser
        driver = webdriver.Chrome(PATH, options=option)
        # Webdriver opens url
        driver.get(URL)
        # Page loads generates full code, source code is then stored as a variable
        soup_source = driver.page_source
        # Variable creates a soup object based on source code in analyzable format
        soup = BeautifulSoup(soup_source, features="lxml")
        # Webdriver closed as beautifulsoup will process remaining code
        driver.quit()
        # Web page's title was a good format so I grabbed it as is
        title = soup.title.text.strip()
        # Element id of HTML code block needed to analyze
        focus = soup.find(id='qlook')
        # Variables to store extracted weather details
        temp = focus.find('div', class_='h2').text.strip()
        rest_info = focus.find_all('p')
        condition = rest_info[0].text.strip()
        description = rest_info[1].text.strip().split('F')
        # Create a formatted string variable to return forecast in speakable form
        feels = 'F' + description[1]
        return_string = f'{title}\n {temp}\nCondition is {condition}\n {feels}'
        return return_string
```

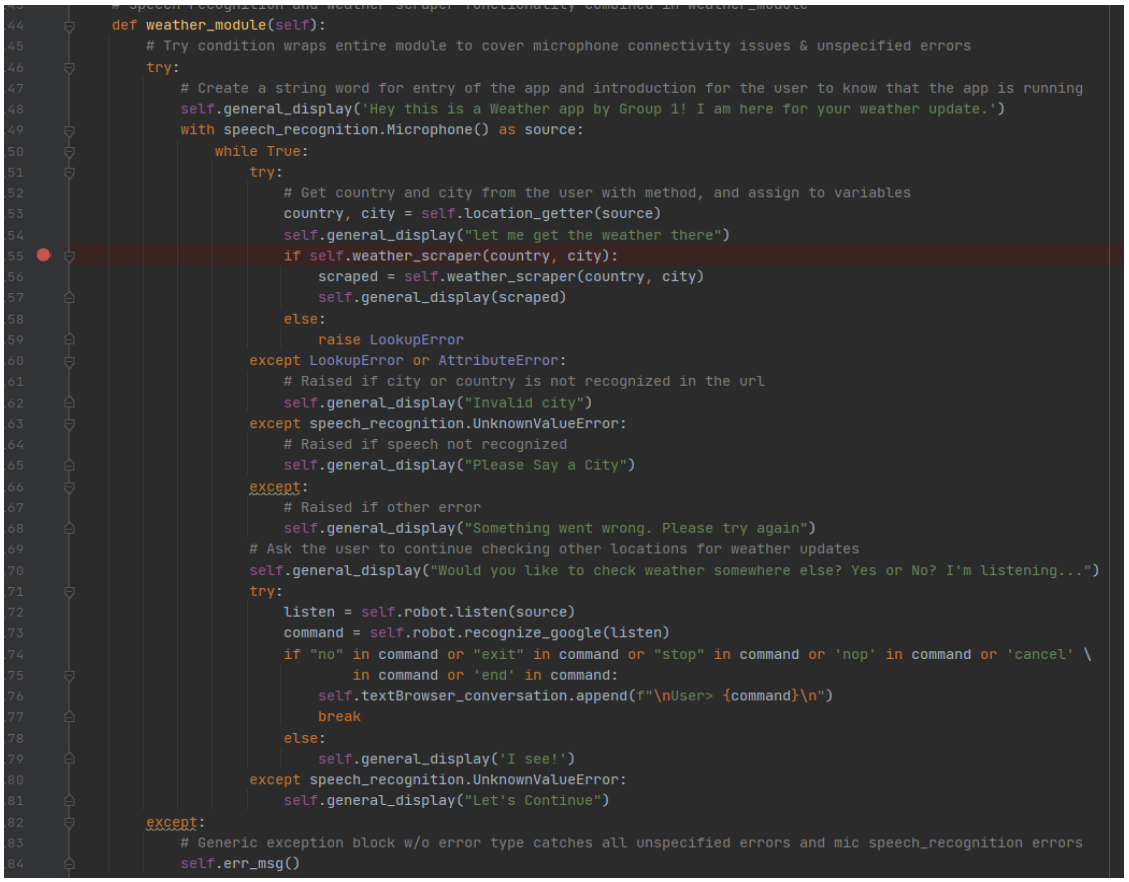
Fig 4: *Time_scraper method scrapes the given website usingBeautifulSoup. The method takes 2 string parameters, city & country, for the URL. Requests library is used to grab the static code for the site, which is very accessible, and BeautifulSoup parses the webpage's code to extract the information we need.*

```
83
84      # Method to statically web scrape time from user input of country & city
85      def time_scraper(self, country: str, city: str):
86          # Create variable to store url with country and city appended which will be opened
87          URL = 'https://www.timeanddate.com/worldclock/' + country + '/' + city
88          # Request gets page's full code
89          page = requests.get(URL)
90          # Variable creates a soup object based on source code in analyzable format
91          soup = BeautifulSoup(page.content, features="lxml")
92          # Web page's title was a good format so I grabbed it as is
93          title = soup.title.text.strip()
94          # Variables to store extracted weather details
95          time = soup.find(id='ct').text.strip()
96          time_zone = soup.find(id='cta').find('a')['title']
97          return f'{title}\n is {time}\nTime Zone is {time_zone}'
98
```

Fig 5: *Time_module integrates speech_recognition function and static web scraper. The speech recognition component is in the location_getter method and gets city & country from the user and returns their values, which are stored in variables. City and country are then sent as parameters to time scraper, with the results being returned in a string. Results are output to GUI and announced.*

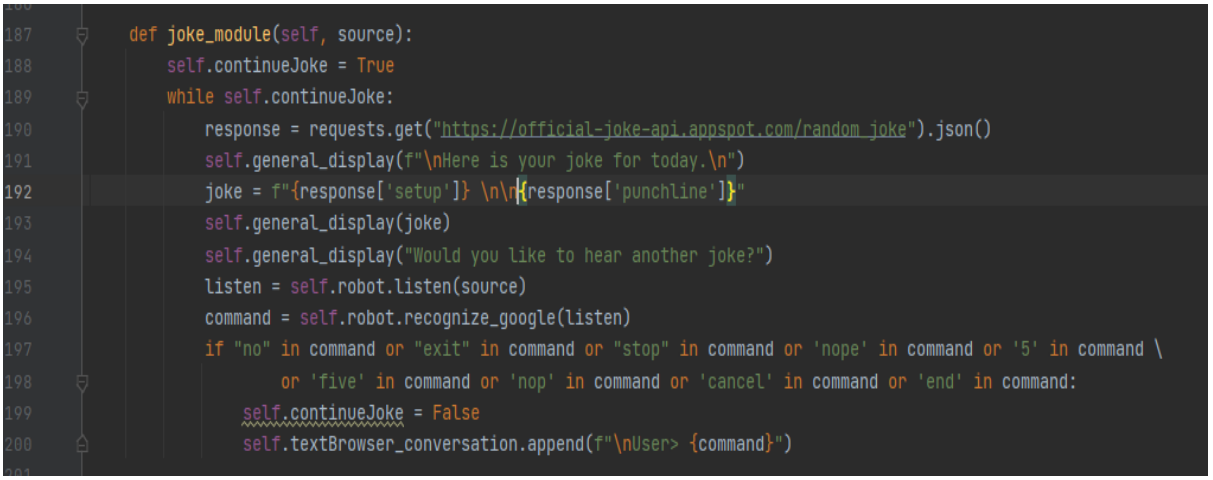
```
101     def time_module(self):
102         # Try condition wraps entire module to cover microphone connectivity issues & unspecified errors
103         try:
104             # Create a string word for entry of the app and introduction for the user to know that the app is running
105             self.general_display('Hey this is a Time app by Group 1! I can tell time anywhere.')
106             with speech_recognition.Microphone() as source:
107                 while True:
108                     try:
109                         # Get country and city from the user with method, and assign to variables
110                         country, city = self.location_getter(source)
111                         self.general_display("let me get the time there")
112                         if self.time_scraper(country, city):
113                             scraped = self.time_scraper(country, city)
114                             self.general_display(scraped)
115                         else:
116                             raise LookupError
117                     except LookupError or AttributeError:
118                         # Raised if city or country is not recognized in the url
119                         self.general_display("Invalid city")
120                     except speech_recognition.UnknownValueError:
121                         # Raised if speech not recognized
122                         self.general_display("Please Say a City")
123                     except:
124                         # Raised if other error
125                         self.general_display("Something went wrong. Please try again")
126
127                     # Ask the user to continue checking other locations for time
128                     self.general_display("Would you like the time of another place? Yes or No? I'm listening...")
129                     try:
130                         listen = self.robot.listen(source)
131                         command = self.robot.recognize_google(listen)
132                         if "no" in command or "exit" in command or "stop" in command or 'nop' in command or 'cancel' in command or 'end' in command:
133                             self.textBrowser_conversation.append(f"\nUser> {command}\n")
134                             break
135                         else:
136                             self.general_display('I see!')
137                     except speech_recognition.UnknownValueError:
138                         self.general_display("Let's Continue")
139         except speech_recognition.UnknownValueError:
140             # Generic exception block without error type catches all unspecified errors & mic speech_recognition errors
141             self.err_msg()
```

Fig 6: *Weather_module integrates speech_recognition function and dynamic web scraper. The speech recognition component is in the location_getter method and gets city & country from the user and returns their values, which are stored in variables. City and country are then sent as parameters to time scraper, with the results being returned in a string. Results are output to GUI and announced.*



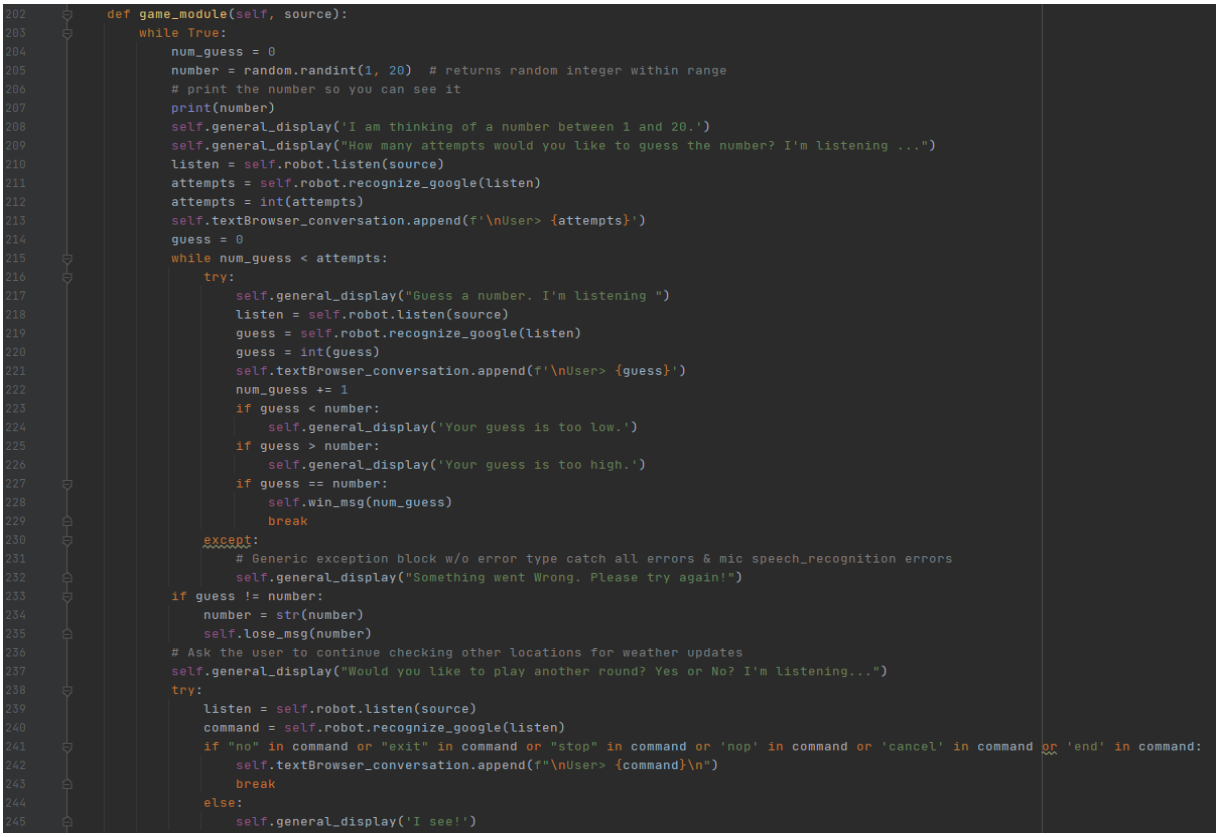
```
    def weather_module(self):
        # Try condition wraps entire module to cover microphone connectivity issues & unspecified errors
        try:
            # Create a string word for entry of the app and introduction for the user to know that the app is running
            self.general_display('Hey this is a Weather app by Group 1! I am here for your weather update.')
            with speech_recognition.Microphone() as source:
                while True:
                    try:
                        # Get country and city from the user with method, and assign to variables
                        country, city = self.location_getter(source)
                        self.general_display("let me get the weather there")
                        if self.weather_scraper(country, city):
                            scraped = self.weather_scraper(country, city)
                            self.general_display(scraped)
                        else:
                            raise LookupError
                    except LookupError or AttributeError:
                        # Raised if city or country is not recognized in the url
                        self.general_display("Invalid city")
                    except speech_recognition.UnknownValueError:
                        # Raised if speech not recognized
                        self.general_display("Please Say a City")
                    except:
                        # Raised if other error
                        self.general_display("Something went wrong. Please try again")
                    # Ask the user to continue checking other locations for weather updates
                    self.general_display("Would you like to check weather somewhere else? Yes or No? I'm listening...")
                    try:
                        listen = self.robot.listen(source)
                        command = self.robot.recognize_google(listen)
                        if "no" in command or "exit" in command or "stop" in command or 'nop' in command or 'cancel' \
                                in command or 'end' in command:
                            self.textBrowser_conversation.append(f"\nUser> {command}\n")
                            break
                        else:
                            self.general_display('I see!')
                    except speech_recognition.UnknownValueError:
                        self.general_display("Let's Continue")
        except:
            # Generic exception block w/o error type catches all unspecified errors and mic speech_recognition errors
            self.err_msg()
```

Fig 7: *Joke Module gets a random joke from an open-source joke API. The request library is used to get the joke and punchline, while json library is used to parse and display the joke.*



```
    def joke_module(self, source):
        self.continueJoke = True
        while self.continueJoke:
            response = requests.get("https://official-joke-api.appspot.com/random_joke").json()
            self.general_display(f"\nHere is your joke for today.\n")
            joke = f"{response['setup']} \n\n{response['punchline']}"
            self.general_display(joke)
            self.general_display("Would you like to hear another joke?")
            listen = self.robot.listen(source)
            command = self.robot.recognize_google(listen)
            if "no" in command or "exit" in command or "stop" in command or 'nope' in command or '5' in command \
                    or 'five' in command or 'nop' in command or 'cancel' in command or 'end' in command:
                self.continueJoke = False
                self.textBrowser_conversation.append(f"\nUser> {command}")
```

Fig 8: *Game Module is a simple guess-the-randomly-generated-number game within the allotted attempts, and integrates taking of user speech from speech recognition.*



```
    def game_module(self, source):
        while True:
            num_guess = 0
            number = random.randint(1, 20)  # returns random integer within range
            # print the number so you can see it
            print(number)
            self.general_display('I am thinking of a number between 1 and 20.')
            self.general_display("How many attempts would you like to guess the number? I'm listening ...")
            listen = self.robot.listen(source)
            attempts = self.robot.recognize_google(listen)
            attempts = int(attempts)
            self.textBrowser_conversation.append(f'\nUser> {attempts}')
            guess = 0
            while num_guess < attempts:
                try:
                    self.general_display("Guess a number. I'm listening ")
                    listen = self.robot.listen(source)
                    guess = self.robot.recognize_google(listen)
                    guess = int(guess)
                    self.textBrowser_conversation.append(f'\nUser> {guess}')
                    num_guess += 1
                    if guess < number:
                        self.general_display('Your guess is too low.')
                    if guess > number:
                        self.general_display('Your guess is too high.')
                    if guess == number:
                        self.win_msg(num_guess)
                        break
                except:
                    # Generic exception block w/o error type catch all errors & mic speech_recognition errors
                    self.general_display("Something went Wrong. Please try again!")
            if guess != number:
                number = str(number)
                self.lose_msg(number)
            # Ask the user to continue checking other locations for weather updates
            self.general_display("Would you like to play another round? Yes or No? I'm listening...")
            try:
                listen = self.robot.listen(source)
                command = self.robot.recognize_google(listen)
                if "no" in command or "exit" in command or "stop" in command or 'nop' in command or 'cancel' in command or 'end' in command:
                    self.textBrowser_conversation.append(f"\nUser> {command}\n")
                    break
                else:
                    self.general_display('I see!')
```

Fig 9: *Menu_selection method allows the user to select the function they would like to use and opens the corresponding module. Method loop will ask the user to continue exploring functionality or exit loop and end app.*

```
    def menu_selection(self):
        self.general_display("What would you like to do today? You can select from options on the left, I'm listening")
        try:
            with speech_recognition.Microphone() as source:
                while self.continueLoop:
                    listen = self.robot.listen(source)
                    command = self.robot.recognize_google(listen)
                    if "no" in command or "exit" in command or "stop" in command or 'nope' in command or '5' in command \
                            or 'five' in command or 'nop' in command or 'cancel' in command or 'end' in command:
                        self.continueLoop = False
                        self.textBrowser_conversation.append(f"\nUser> {command}")
                        self.exit_msg()
                    elif "weather" in command or "get the weather" in command or "1" in command or "one" in command:
                        self.continueLoop = True
                        self.general_display("You selected to get the weather...")
                        self.weather_module()
                    elif "time" in command or "get the time" in command or "3" in command or "three" in command:
                        self.continueLoop = True
                        self.general_display("You selected to get the time...")
                        self.time_module()
                    elif "game" in command or "play a game" in command or "play" in command or "2" in command or "two" in command:
                        self.continueLoop = True
                        self.general_display("You selected to play a game...")
                        self.game_module(source)
                    elif "joke" in command or "tell me a joke" in command or "please tell me a joke" in command or "4" in command or "four" in command:
                        self.joke_module(source)
                    else:
                        self.continueLoop = True
                        self.general_display('I did not understand so I will tell you a joke!')
                        self.joke_module()

                    self.stop_event2()
        except:
            # Generic exception block without error type catches all unspecified errors and mic speech recognition errors
            self.err_msg()
```

Fig 10: *2 Methods for controlling end of loops. Stop_event2 method is called to determine whether to exit the application or continue exploring modules.*

```
246
247     def stop_event2(self):
248         # self.stop_event2()
249         self.general_display("Would you like to continue? Yes or No? I'm listening...")
250         try:
251             with speech_recognition.Microphone() as source:
252                 listen = self.robot.listen(source)
253                 command = self.robot.recognize_google(listen)
254                 if "no" in command or "exit" in command or "stop" in command or 'nop' in command or 'cancel' in command or 'end' in command:
255                     self.continueLoop = False
256                     self.textBrowser_conversation.append(f"\nUser> {command}")
257                     # Closing message
258                     self.exit_msg()
259                 else:
260                     self.continueLoop = True
261                     self.general_display("Let's continue! You can select from options on the left, I'm listening")
262         except:
263             # Generic exception block w/o error type catches all unspecified errors and mic speech_recognition errors
264             self.err_msg()
265
266     def stop_event(self):
267         if self.continueLoop:
268             self.continueLoop = False
269
```

Fig 11: *Location_getter method uses speech recognition to get user input of city and country and returns them as 2 string variables.*

```
307     def location_getter(self, source):
308         while True:
309             self.general_display('What country are you looking for? Listening...')
310             # Try condition for invalid country and unrecognized speech exceptions
311             try:
312                 listen = self.robot.listen(source)
313                 # listen to the voice and take the first word if there are many
314                 place = self.robot.recognize_google(listen).lower().split(' ')
315                 country = '-'.join(place)
316                 self.textBrowser_conversation.append(f"\nUser> {place}")
317                 # If statement validates input country with pycountry module
318                 if pycountry.countries.search_fuzzy(country):
319                     # Url takes america as usa, so this if condition check for that
320                     if pycountry.countries.search_fuzzy(country) == pycountry.countries.search_fuzzy("America"):
321                         country = "usa"
322                     break
323                 else:
324                     # Raise error if country is not validated
325                     raise LookupError
326             except LookupError or AttributeError:
327                 # Raised if country not validated
328                 self.general_display("Invalid Country")
329             except speech_recognition.UnknownValueError:
330                 # Raised if speech not recognized
331                 self.general_display("Please Say a Country")
332         # Get city from the user
333         while True:
334             self.general_display("What city are you looking for? I'm listening...")
335             # Try condition for invalid city and unrecognized speech exceptions
336             try:
337                 listen = self.robot.listen(source)
338                 # listen to the voice and take the first word if there are many
339                 place = self.robot.recognize_google(listen).lower().split(' ')
340                 self.textBrowser_conversation.append(f"\nUser> {place}")
341                 city = '-'.join(place)
342                 self.general_display(city + ", " + country)
343                 break
344             except:
345                 # Raised if speech not recognized
346                 self.general_display("Please Say a City")
347
348         return country, city
349
```

Fig 12: *Methods return string message outputs to GUI and speech-to-text.*

```
def general_display(self, msg: str):
    # Since forecast results have successfully been returned say forecast and break loop
    self.textBrowser_conversation.append(f"\nComputer> {msg}\n")
    self.ROBOT_VOICE(msg)
    # print(msg)

def exit_msg(self):
    # Closing message
    self.general_display('Thank you. Talk to you again soon!')
    self.textBrowser_conversation.append(f"""
                                """)
```

Fig 13: *Methods return string message output to GUI and speech-to-text, on critical error.*

```
def win_msg(self, num_guess):
    self.general_display('Good job! You guessed my number in ' + str(num_guess) + ' tries.')
    self.textBrowser_conversation.append("""
                                """)

def lose_msg(self, number):
    self.general_display('Sorry, you ran out of attempts. I was thinking of the number ' + number)
    self.textBrowser_conversation.append("""
                                """)
```

Fig 13: *Error message method that output error message in GUI and speech. Driver Code runs the application.*

Fig 14: *GUI display on startup of application.*

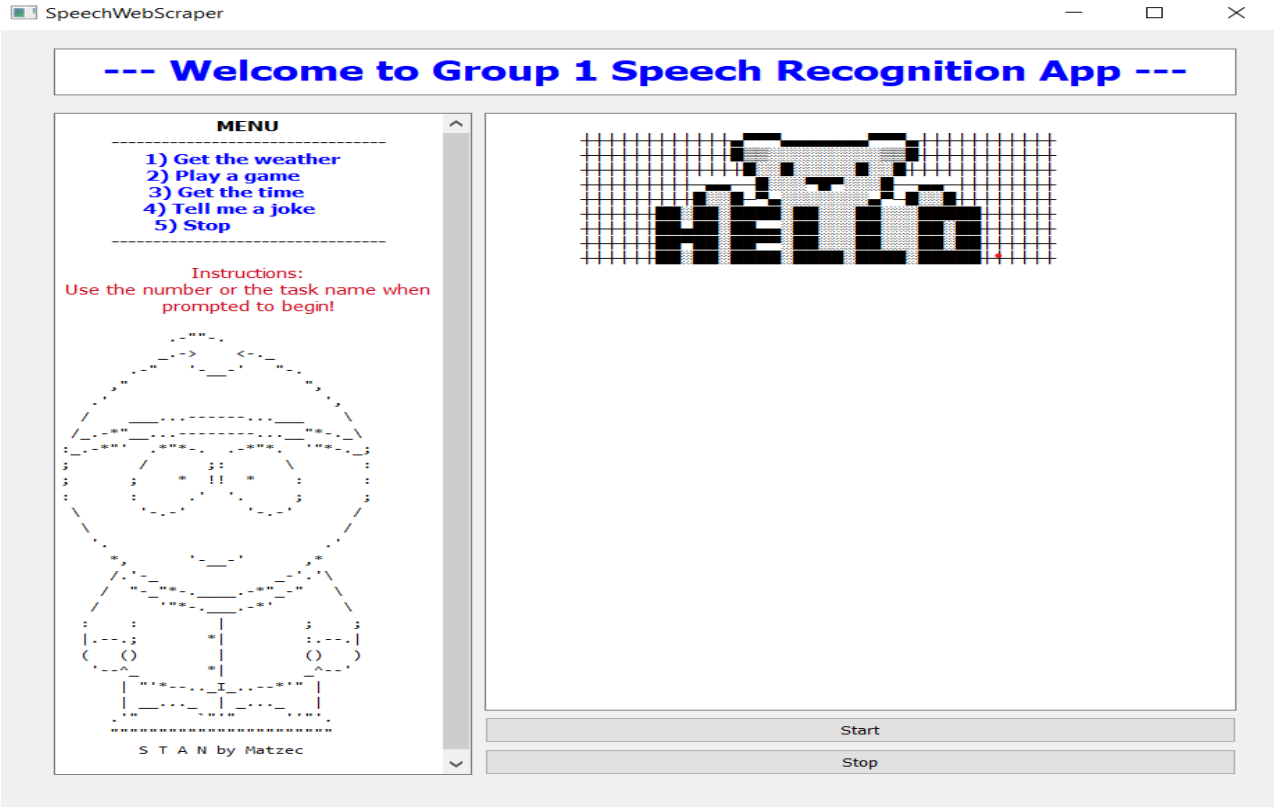


Fig 15: *Operation of time module. GUI opens and displays all text. In this screen, the user selects the time module, and is asked and provides a country and city. The user is then given the time VIA speech-to-text, & it is displayed in GUI. User selecting not to continue, exits module.*

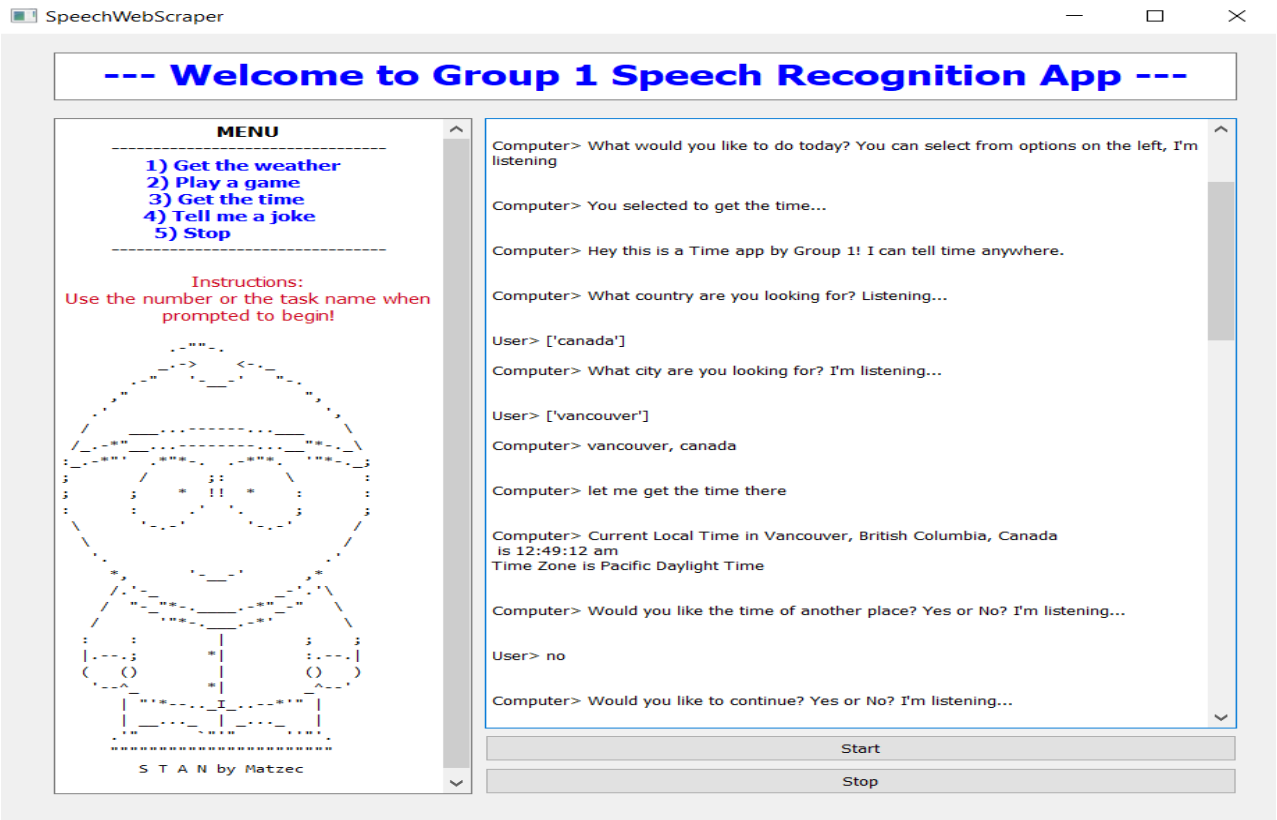


Fig 15: *Operation of weather module. GUI opens and displays all text. In this screen, the user selects the weather module, and is asked and provides a country and city. The user is then given the weather forecast VIA speech-to-text, & it is displayed in GUI. User selecting not to continue, exits module.*

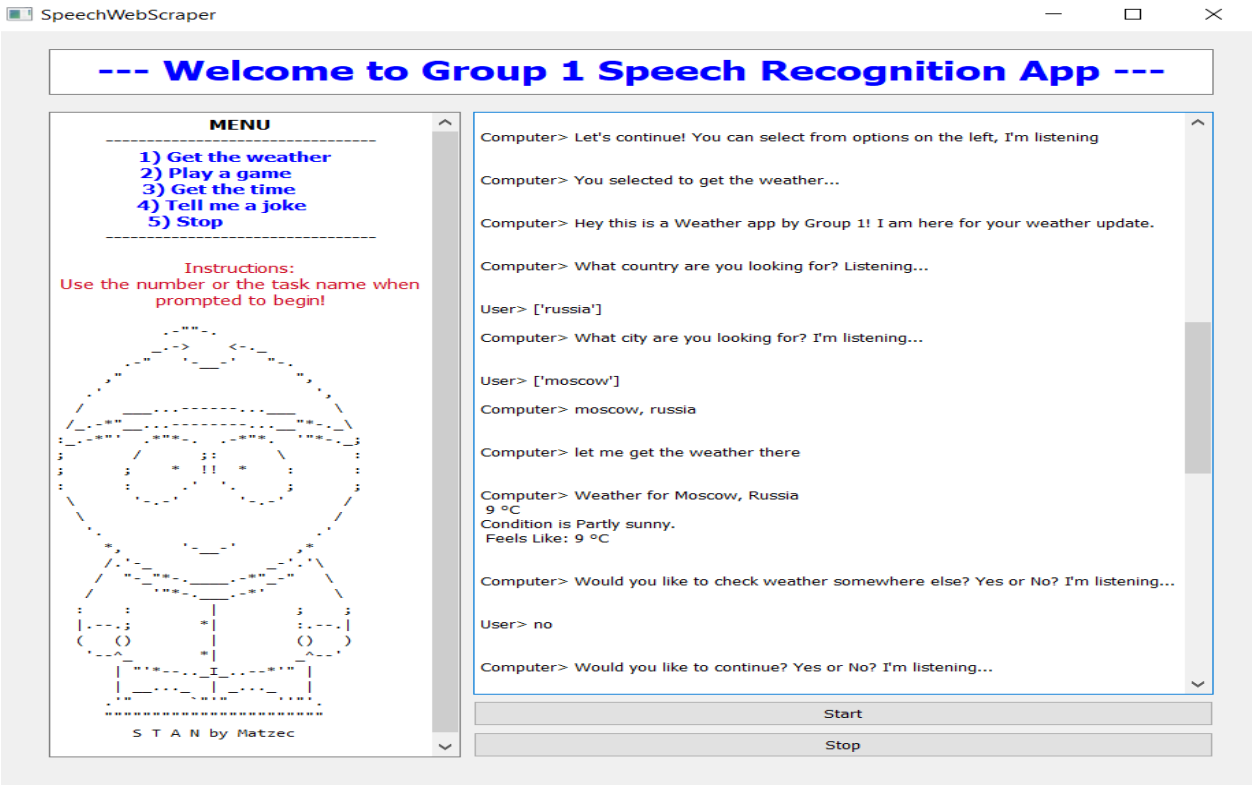


Fig 16: *Operation of joke module. GUI opens and displays all text. In this screen, the user selects the joke module, and is told a random joke VIA speech-to-text, & on the GUI. User selecting not to continue, exits module.*

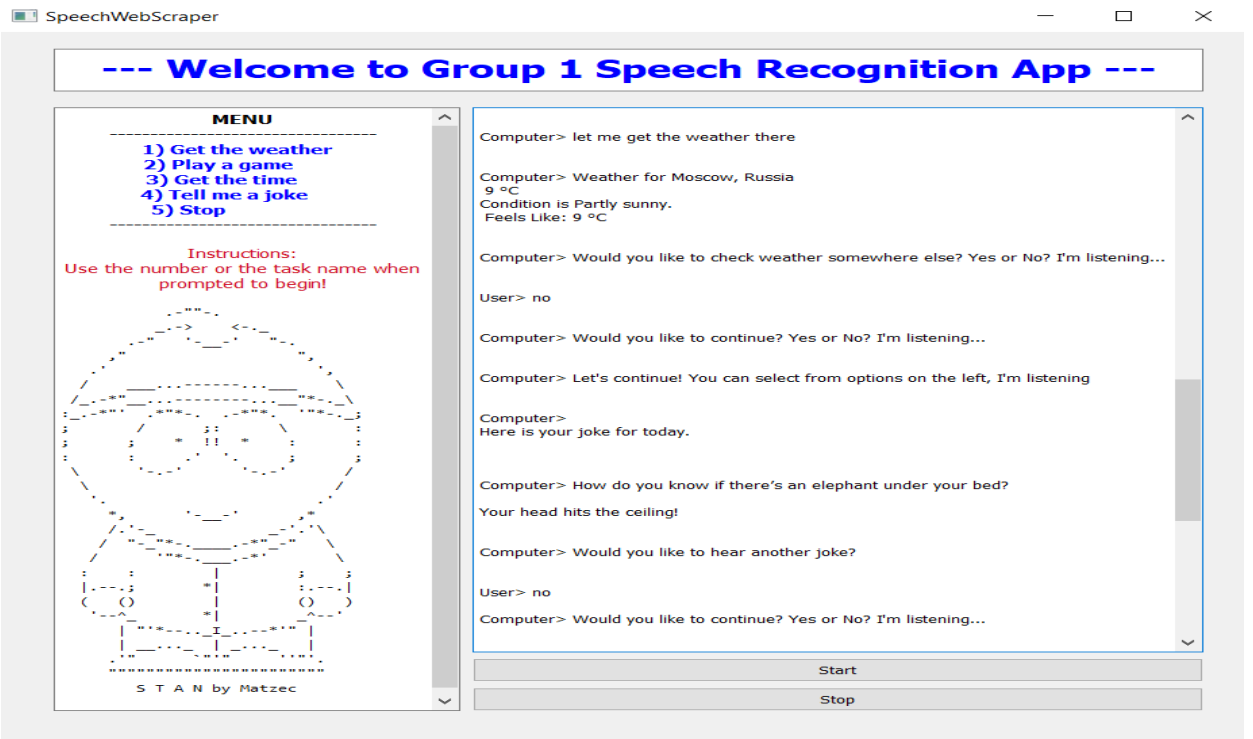


Fig 17: *Operation of game module. In these screens, the user is asked for the number attempts they would like to take. Then the user has that number of attempts to guess a randomly generated number between 1-20. Victory or defeat, results are output to the user VIA speech-to-text, & on the GUI. User selecting not to continue, exits module.*

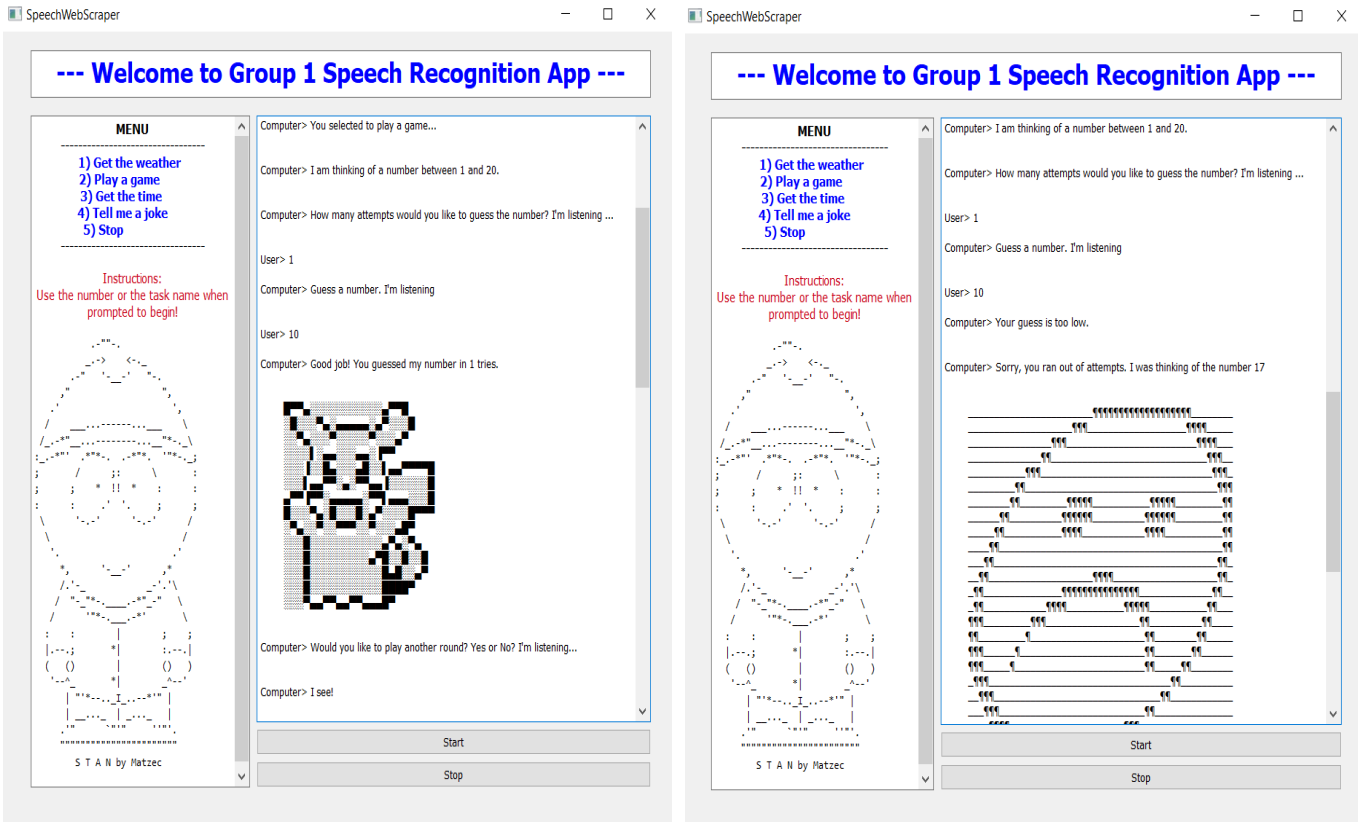
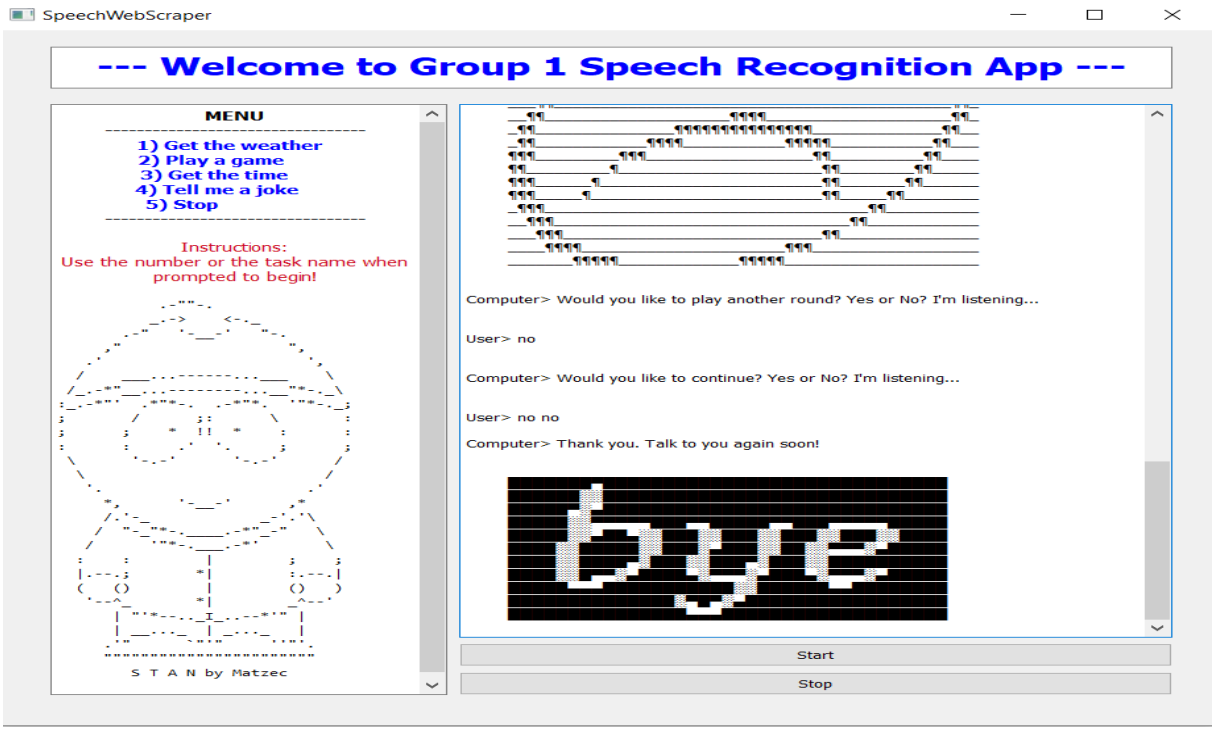


Fig 18:.*Exit the application by choosing not to continue on to another module.*



Conclusions and Future Work

There are several things we learn from creating our voice command application. We learn how to use Python speech recognition packages to implement voice commands and connect it using an API to retrieve weather data and output the result back to the user. We also made use of another Python package, pyttsx3, to convert text to audio. There are many ways that we can take this project further. Depending on time, we plan to implement several functions, one of them being able to fetch Google calendar data, where a user can set appointments and reminders. Another addition we will be including in the future is a chat room using sockets that will allow users to participate in private chat room conversations with the use of speech recognition. More functionality can be added by integration with Google Maps, Facebook, Uber and other services and platforms

Bibliography

Retrieved from https://www.w3schools.com/python/

Barry, P. (2017). *Head first Python*. O'Reilly.

RAMALHO, L. (2021). *FLUENT PYTHON: Clear, concise, and effective programming*. O'REILLY MEDIA, INC, USA.

Welcome to Python.org. (n.d.). Retrieved from https://www.python.org/doc/port.