

This file will provide a step by step explanation of how I was able to make my voice assistant. To begin my project, I installed pip, the package installer for Python. I used the following YouTube video to assist me in doing so:

📺 6 PYTHON How to install PIP LATEST on macbook using terminal

Then I started by importing the following modules necessary for the Python file to run in the terminal. (A terminal on a Mac is equivalent to a command prompt window for Windows computers.)

- gTTS
- playsound
- SpeechRecognition
- pyaudio

However, I couldn't install pyaudio through the terminal and repeatedly got the following error message:

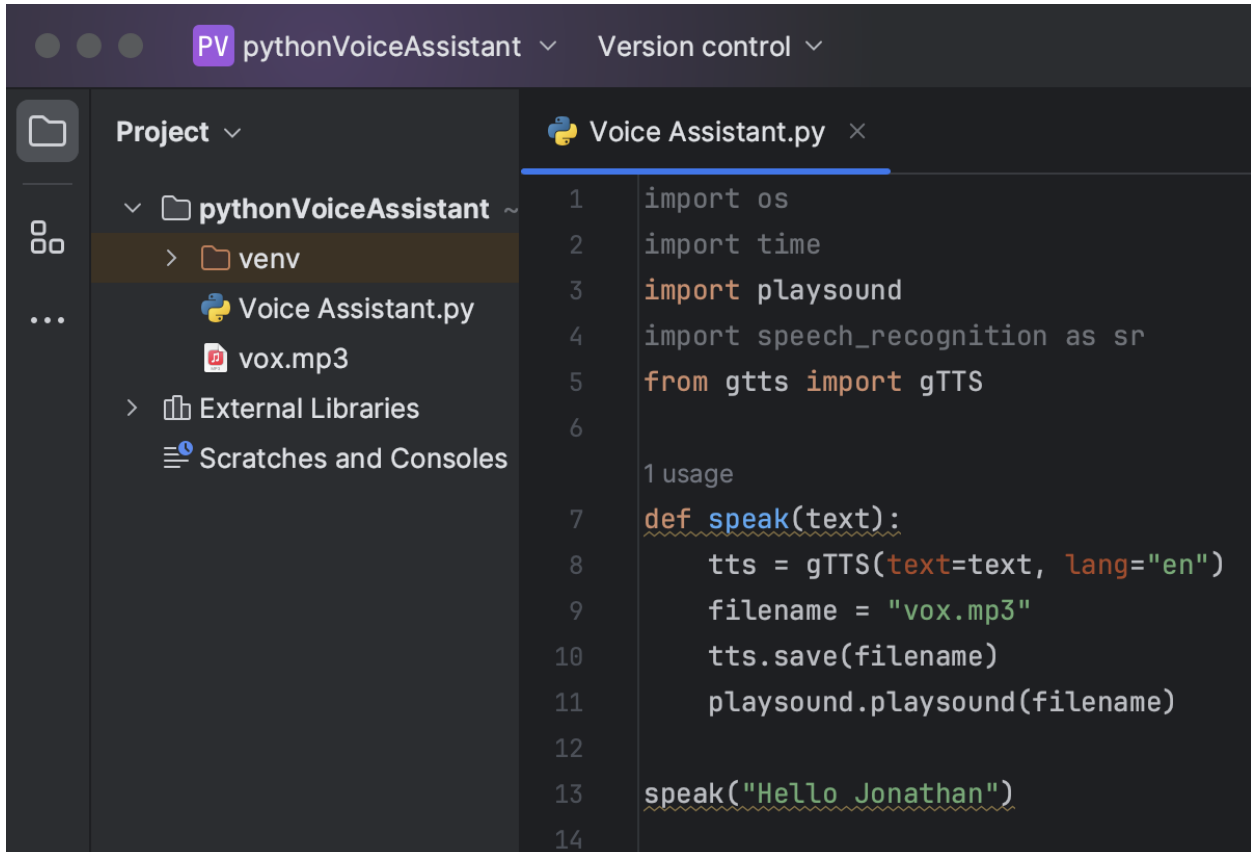
**note:** This error originates from a subprocess, and is likely not a problem with pip.

**error:** subprocess-exited-with-error

Installing pyaudio was essential if I wanted to be able to communicate with Vox, so it could not be ignored. I did some research and found out that I can install a prerequisite, portaudio that can be found after installing brew through this link: <https://brew.sh/>

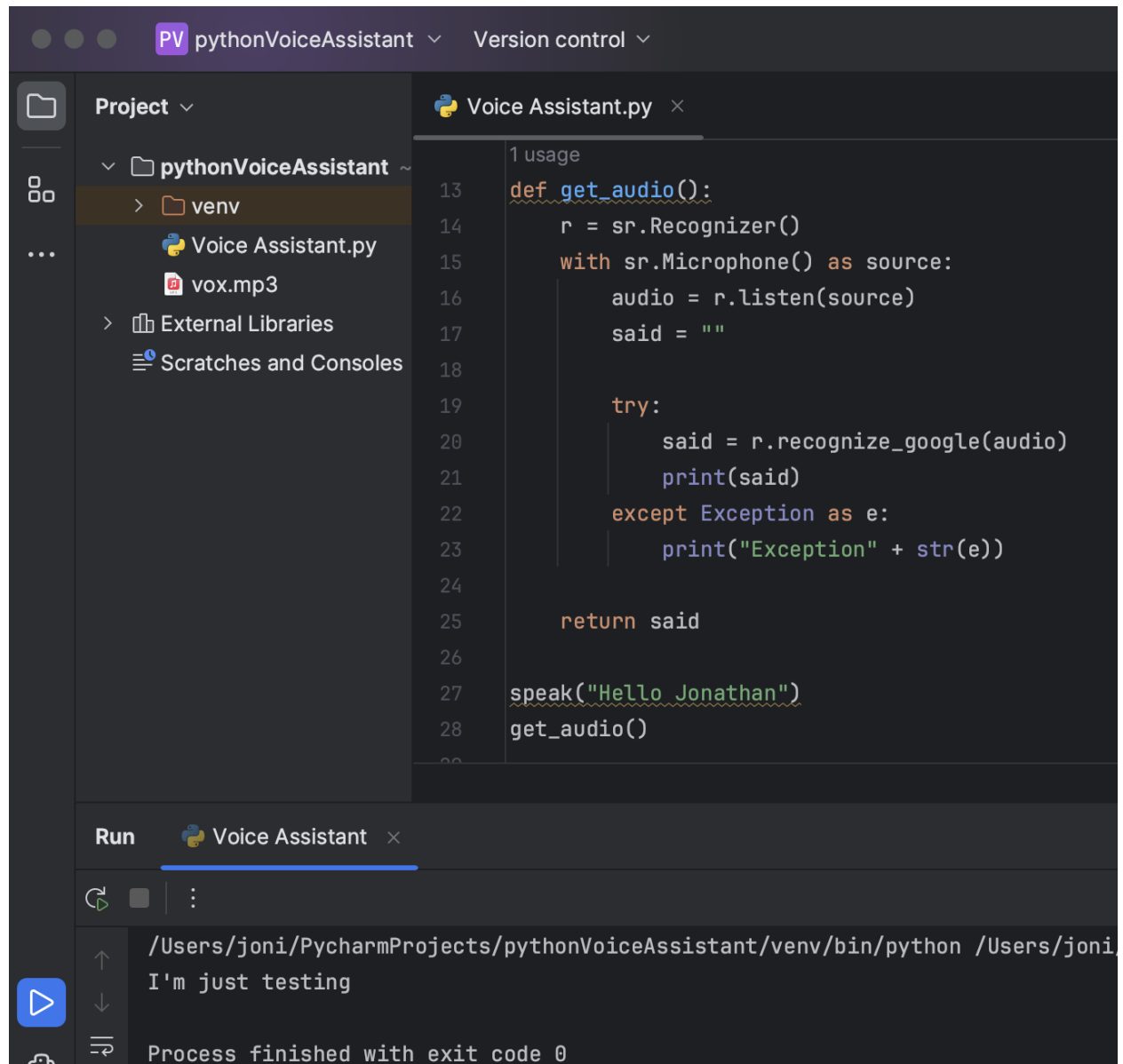
But I still couldn't install pyaudio using the terminal. I decided to use the PyCharm terminal instead, believing it was a problem with the Macbook terminal, and it worked. I finally successfully installed pyaudio.

Now it's time to begin programming. The goal now is to ensure that some of these modules are indeed working, and we can test this out by prompting Vox to read a text message, denoting that the program can provide a voice output through the speakers before proceeding to receiving voice inputs. The following code does so and runs smoothly; however, there is a one second delay before Vox starts speaking the written text in the speak function, "Hello Jonathan".



```
1 import os
2 import time
3 import playsound
4 import speech_recognition as sr
5 from gtts import gTTS
6
7 def speak(text):
8     tts = gTTS(text=text, lang="en")
9     filename = "vox.mp3"
10    tts.save(filename)
11    playsound.playsound(filename)
12
13    speak("Hello Jonathan")
14
```

Now that we have allowed Vox to output our desired texts, we will proceed to make Vox able to interpret spoken input. For this, it is essential that we have a clear and working microphone. We will test this by checking if Vox can return the spoken words using texts as an output. The following code allows us to speak to the user, no matter how long the sentence. If two seconds go by without the user speaking, Vox assumes the user has completed their sentence and provides a text output of what the user said. At this stage, the only issue I have with Vox is the delay before the user starts speaking (a second after Vox says hello). In the output below, I said, "I'm just testing," but Vox was able to interpret the second half because of the one second delay.



The screenshot shows the PyCharm IDE interface. The top toolbar includes a 'Run' button (a green play icon) and a 'Debug' button (a blue play icon). The left sidebar displays the project structure for 'pythonVoiceAssistant', showing a 'venv' folder and files 'Voice Assistant.py' and 'vox.mp3'. The main editor window shows the code for 'Voice Assistant.py'. The code defines a 'usage' function, a 'get\_audio()' function that uses 'sr.Recognizer()' and 'sr.Microphone()' to listen and recognize speech, and a 'speak()' function. The 'main()' function calls 'speak("Hello Jonathan")' and then 'get\_audio()'. The bottom panel shows the 'Run' output, indicating the command executed and the output 'I'm just testing', followed by the message 'Process finished with exit code 0'.

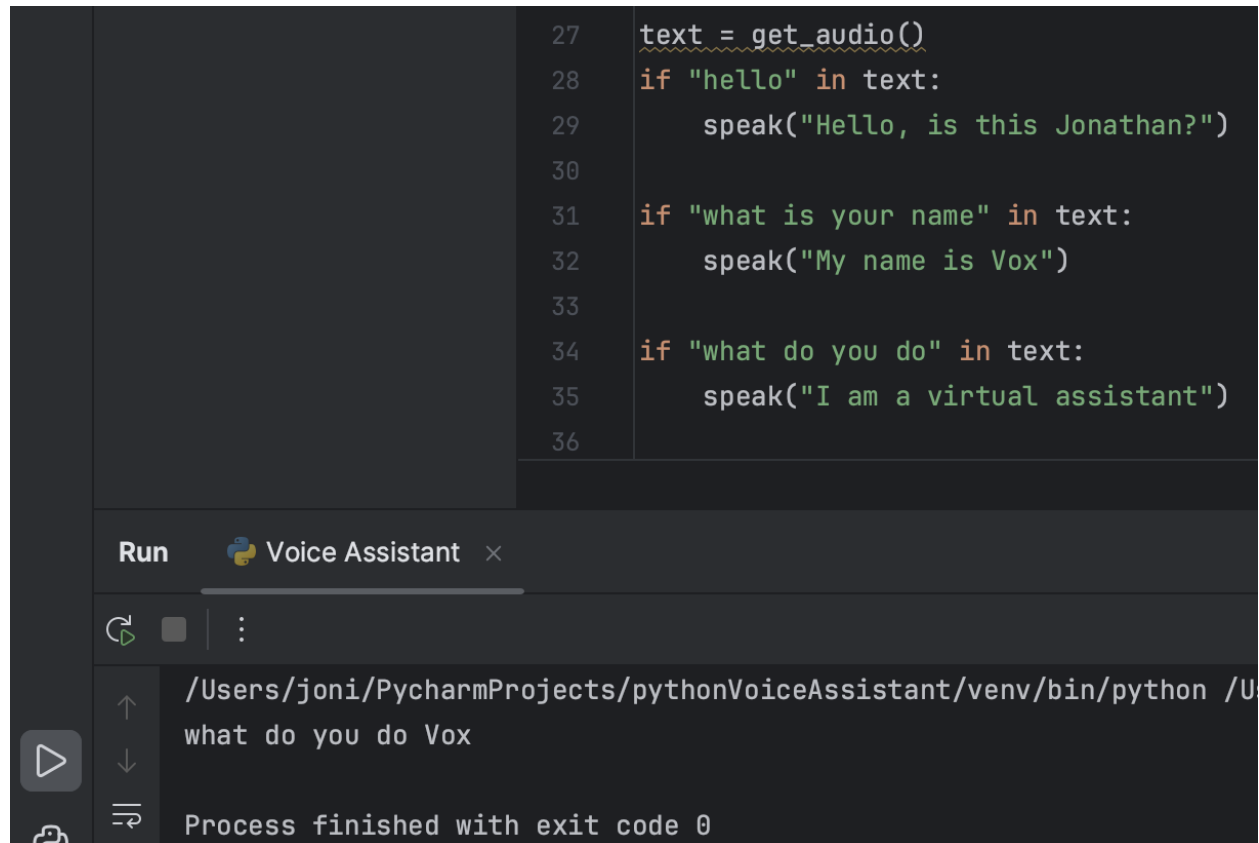
```
1 usage
13 def get_audio():
14     r = sr.Recognizer()
15     with sr.Microphone() as source:
16         audio = r.listen(source)
17         said = ""
18
19     try:
20         said = r.recognize_google(audio)
21         print(said)
22     except Exception as e:
23         print("Exception" + str(e))
24
25     return said
26
27 speak("Hello Jonathan")
28 get_audio()
```

Run Voice Assistant ×

/Users/joni/PycharmProjects/pythonVoiceAssistant/venv/bin/python /Users/joni  
I'm just testing

Process finished with exit code 0

This is great progress, as now there is a back and fourth interaction between the user and Vox; however, it is either Vox speaking the text or interpreting the speech. We need to make it a verbal interaction between the user and Vox, and this can be done using simple if statements. By using these conditions, we are restricting the output to the response we desire. In the example below, I asked Vox what it does, and Vox responded, "I am a virtual assistant."



```
27 text = get_audio()
28 if "hello" in text:
29     speak("Hello, is this Jonathan?")
30
31 if "what is your name" in text:
32     speak("My name is Vox")
33
34 if "what do you do" in text:
35     speak("I am a virtual assistant")
36
```

Run Voice Assistant x

what do you do Vox

Process finished with exit code 0

The next step was to start implementing features to automate tasks. A great place to start was to enable the Google Calendar API through the following link:

<https://developers.google.com/calendar/api/quickstart/python>

The following YouTube video (in the first 6 minutes) [Google Calendar Automation in Python](#) provides a step by step guide on how to enable Google Calendar to perform automation in our Python project. Firstly, we need to download the respective json credential and paste it in our project folder. Then, we install the pip modules on the terminal to use the respective pip packages. Following the installation, we can simply copy and paste the given code Configuring the ample', remove some of the bugs, and refactor some portions of the code to fit our project specifically. We run the code, which redirects us to an email sign in page where we need to allow the API to view our calendar in order to inform us about our upcoming events. This will automatically generate a pickle or a json file in our project folder under the token name, which stores our credential information. We conclude by testing if the code successfully returns dates and times for the events to happen. The output should look similar to the screenshot below:

```
1 usage
def get_events(n, service):
    # n is the amount of events we want to get and service is the service returned to us from the auth
    # Call the Calendar API
    now = datetime.datetime.utcnow().isoformat() + 'Z' # 'Z' indicates UTC time
    print(f'Getting the upcoming {n} events')
    events_result = service.events().list(calendarId='primary', timeMin=now,
                                          maxResults=n, singleEvents=True,
                                          orderBy='startTime').execute()
    events = events_result.get('items', [])

    if not events:
        print('No upcoming events found.')
        # return

    # Prints the start and name of the next 10 events
    for event in events:
        start = event['start'].get('dateTime', event['start'].get('date'))
        print(start, event['summary'])

service = authenticate_googleCalendar()
get_events(n=2, service)
```

Run Voice Assistant x

```
/Users/joni/PycharmProjects/pythonVoiceAssistant/venv/bin/python /Users/joni/PycharmProjects/pythonVoiceAssistant/Voice Assistant.py
Getting the upcoming 2 events
2023-11-20T11:15:00+03:00 CSCI 135 Group Project Presentation
2023-11-20T12:30:00+03:00 CSCI 135 Personal Project Presentation
Process finished with exit code 0
```

pythonVoiceAssistant > Voice Assistant.py 90:23 LF UTF-8 4 s

Now we get into how a user may request an activity for a specific date. We begin this by initializing the different ways the user can say this. For example, they can say, “What do I have scheduled for Monday?” or they can even ask, “Do I have any plans for the 5th of next month?” and Vox should be able to interpret this and go through the calendar events from Google Calendar to provide that date as the response. In order to do this, we need to set up a function that gets the requested date and returns it according to the user's request. Finally, when we ask Vox a question, like I did in the screenshot below, Vox provides a written response with the date. If the user says something that doesn't make sense, Vox simply returns “none.” This proves that our virtual assistant can understand voice inputs for dates, so we may proceed to actually utilize it to automate tasks dealing with date scheduling.

The screenshot shows the PyCharm IDE interface. The top bar indicates the project is 'pythonVoiceAssistant' and 'Version control' is active. The left sidebar shows the 'Project Files' view with the following structure:

- ~/PycharmProjects/pythonVoiceAssistant
  - .idea
  - venv
    - credentials.json
    - token.json
  - Voice Assistant.py
  - vox.mp3

The main editor displays the 'Voice Assistant.py' file with the following code:

```
135 if month == -1 and day != -1: # if we didn't find a month,
136     if day < today.day:
137         month = today.month + 1
138     else:
139         month = today.month
140
141 # if we only found a dta of the week
142 if month == -1 and day == -1 and day_of_week != -1:
143     current_day_of_week = today.weekday()
144     # Find the difference between the two dates
145     dif = day_of_week - current_day_of_week
146     # So that we can start fresh every new week/every Monday
147     if dif < 0:
148         dif += 7
149         if text.count("next") >= 1:
150             dif += 7
151
152     return today + datetime.timedelta(dif)
153
154 if day != -1:
155     return datetime.date(month=month, day=day, year=year)
156
157 text = get_audio()
158 print(get_date(text))
```

The bottom panel shows the 'Run' output for 'Voice Assistant'. The command executed is:

```
/Users/joni/PycharmProjects/pythonVoiceAssistant/venv/bin/python /Users/joni/PycharmProjects/p
```

The output shows the user's input and the program's response:

```
do I have plans on November 20th
2023-11-20
```

The process finished with exit code 0.

Now the next step is to modify the “get\_events()” function so that we can give it a date that we get from our “get\_date()” function. These functions will work side by side so that our “get\_events()” function will return the list of events scheduled on a specific date. To do this, we need to get our date in a UTC (Coordinated Universal Time or Universal Time Coordinated) time format. We need to install, import, and utilize the ‘pytz’ module for this. Finally, we test if the “get\_events()” function can actually get the event from a given date, and we get an output like:

pythonVoiceAssistant Version control

Project Files

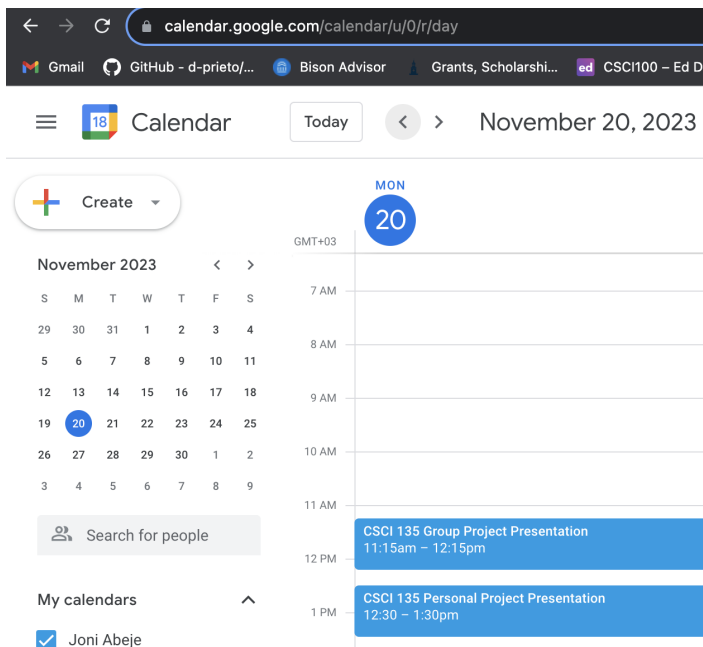
- ~/PycharmProjects/pythonVoiceAssistant
  - .idea
  - venv
    - credentials.json
    - token.json
    - Voice Assistant.py
    - vox.mp3

Voice Assistant.py

```
66 with open('token.json', 'w') as token:
67     token.write(creds.to_json())
68
69
70 service = build(serviceName='calendar', version='v3', credentials=creds)
71
72 return service
73
74 1 usage
75 def get_events(day, service):
76     # Call the Calendar API
77     date = datetime.datetime.combine(day, datetime.datetime.min.time())
78     end = datetime.datetime.combine(day, datetime.datetime.max.time())
79     utc = pytz.UTC
80     date = date.astimezone(utc)
81     end = end.astimezone(utc)
82     events_result = service.events().list(calendarId='primary', timeMin=date.isoformat(),
83                                         timeMax=end.isoformat(), singleEvents=True,
84                                         orderBy='startTime').execute()
85     events = events_result.get('items', [])
86
87     if not events:
88         print('No upcoming events found.')
89         # return
90
91 get_events() > if not events
```

Run Voice Assistant

/Users/joni/PycharmProjects/pythonVoiceAssistant/venv/bin/python /Users/joni/PycharmProjects/pythonVoiceAssistant/Voice Assistant.py  
what events do I have on Monday  
2023-11-20T11:15:00+03:00 CSCI 135 Group Project Presentation  
2023-11-20T12:30:00+03:00 CSCI 135 Personal Project Presentation  
Process finished with exit code 0



Now that we have confirmed that Vox is capable of interpreting date requests and returning the corresponding events from a voice input, we will move on to getting a verbal output from Vox instead of a text response. We include things like spacing between our date and time and adding am or pm to the time for a clearer response from Vox. Additionally, we're currently calling the "get\_events()" and "get\_date()" functions every time we speak to our voice assistant. We need to restrict it so that Vox only responds with events when we say something related to a date. To do so, we create a list of phrases that Vox can detect, such as "What do I have on...", "When is...", "What is happening on...", "Do I have any plans on...", "Am I free on...", "Am I busy on...", and so on, to be able to call the "get\_events()" function. Finally, we loop through these phrases to provide a sense of direction for what we are requesting Vox to do.

The screenshot shows an IDE window titled "pythonVoiceAssistant" with a "Version control" dropdown. The left sidebar shows the "Project Files" tree with the following structure:

- ~/PycharmProjects/pythonVoiceAssistant
  - .idea
  - venv
    - credentials.json
    - token.json
    - Voice Assistant.py
    - vox.mp3

The main editor displays the code in "Voice Assistant.py" with line numbers 168 to 185. The code is as follows:

```

168     if day != -1:
169         return datetime.date(month=month, day=day, year=year)
170
171     SERVICE = authenticate_googleCalendar()
172     print("Start speaking")
173     text = get_audio()
174
175     CALENDAR_STRS = ["what do i have", "do i have plans", "am i busy", "what is happening"
176                     "do I have any plans on", "am I free", "when is"]
177
178     for phrase in CALENDAR_STRS:
179         # Allow Vox to interpret all texts, typically with upper case letters like I and th
180         if phrase in text.lower():
181             date = get_date(text)
182             if date:
183                 get_events(date, SERVICE)
184             else:
185                 speak("Please Try Again")

```

The bottom panel shows the "Run" output for "Voice Assistant". The output is as follows:

```

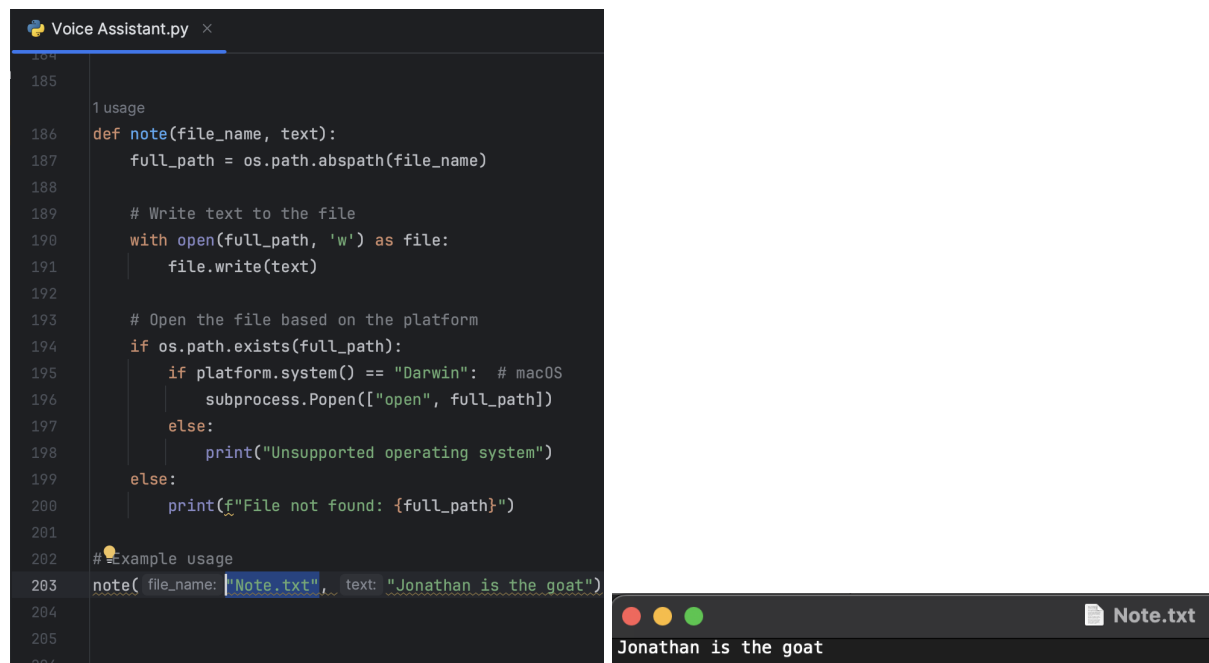
/Users/joni/PycharmProjects/pythonVoiceAssistant/venv/bin/python /Users/joni/PycharmProjects/pythonVoiceAssistant/Vo
Start speaking
what do I have on November 20th
2023-11-20T13:30:00+03:00 CSCI 135 Group Project Presentation
2023-11-20T14:00:00+03:00 CSCI 135 Personal Project Presentation
2023-11-20T14:30:00+03:00 Math tutoring
Process finished with exit code 0

```

The bottom status bar shows "pythonVoiceAssistant" and "Voice Assistant.py" with a timestamp of "185:38".



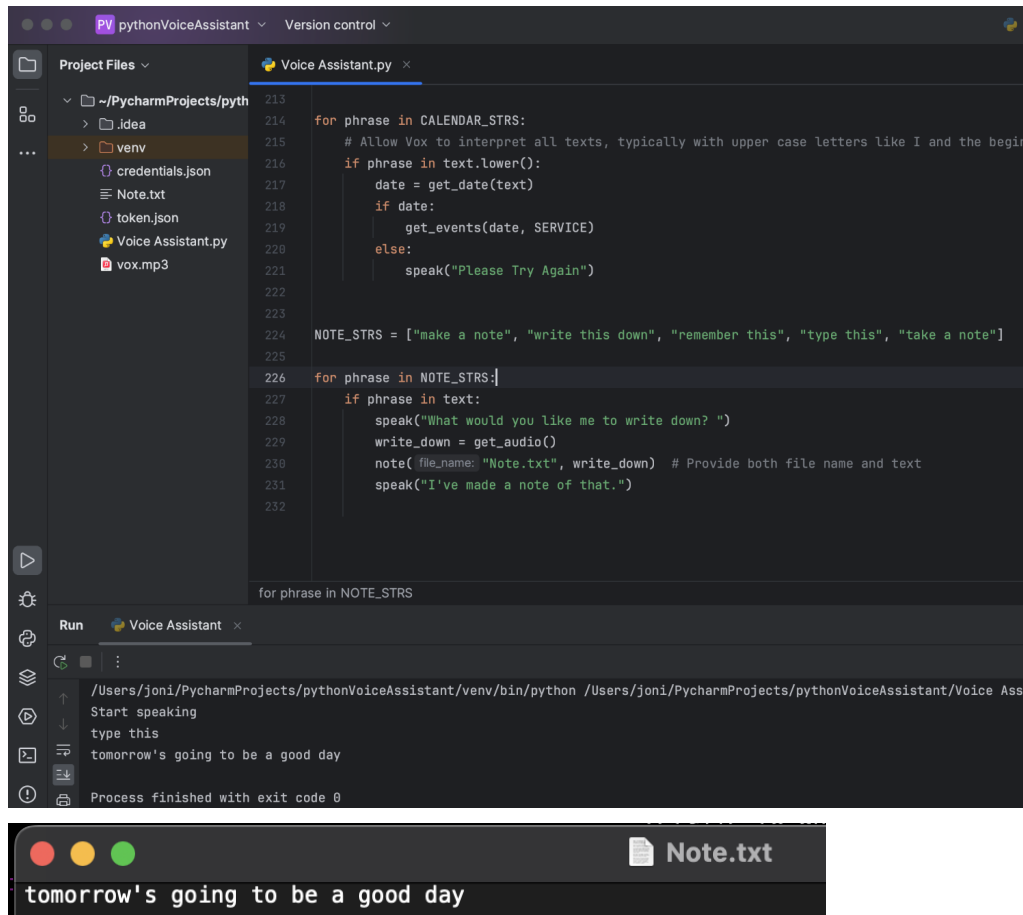
We are completely done with automating tasks regarding dates. We have allowed Vox to tell us about any event we have on any date we ask. Another task automation we can do using our voice assistant is to program Vox to open applications or programs for us. To do this, we begin by importing the 'subprocess' module. For our specific case, we will make our voice assistant open our TextEditor app so that it can make notes. The following code triggers PyCharm to open up the TextEditor note saved "Note.txt" with the text inside the note reading, "Jonathan is the goat."



```
184
185
186 1 usage
187 def note(file_name, text):
188     full_path = os.path.abspath(file_name)
189
190     # Write text to the file
191     with open(full_path, 'w') as file:
192         file.write(text)
193
194     # Open the file based on the platform
195     if os.path.exists(full_path):
196         if platform.system() == "Darwin": # macOS
197             subprocess.Popen(["open", full_path])
198         else:
199             print("Unsupported operating system")
200     else:
201         print(f"File not found: {full_path}")
202
203 # Example usage
204 note(file_name="Note.txt", text="Jonathan is the goat")
205
206
```

Jonathan is the goat

Since we want a spoken output, we do something similar to our previous calendar code for returning events in terms of prompting it to respond only when requested. This time, the training phrases would be things like: "Make a note", "Take a note", "Write this down", "Remember this", "Type this", "Go to notes, and so on. The following code is responsible for providing the following output but also prompting whatever was printed to appear in the Notes.txt file too.



These are just a few things we could do with our voice assistant to make our lives easier. Other things we could use Vox for to automate tasks include writing an email, opening up a YouTube video and even leaving a comment, attending a Google Meet event through our commands, and much more. One thing I look forward to doing in the future is making purchases on websites through my personal voice assistant. Of course, that would be significantly more difficult to program, but we have already achieved so much, so I'm confident in my abilities to do that.