Odd Semester (2024)



**BINUS UNIVERSITY**



**BINUS INTERNATIONAL**



**Assignment Cover Letter**

**(Individual Work** **)**

**Student Information :**

**Surname**  **Given Names Student ID Number**

1. Herdito  Karel Bondan Andoro 2440032373

**Course Name :** Program Design Methods

**Name of Lecturer(s) :** Jude Joseph Lamug Martinez

**Course Code**  **:** COMP6056

**Class**  **:** L1AC

**Major**   **:** Computer Science

**Title of Assignment** **:** Wolf Virtual Assistant

**Type of Assignment** **:** Final Project

**Submission Pattern :**

**Due Date**  **:** 14-01-2021   **Submission Date**  **:**  08-01-2021

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer’s instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

**Plagiarism/Cheating**

BiNus International seriously regards all forms of plagiarism, cheating and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

**Declaration of Originality**

By signing this assignment, I understand, accept and consent to BiNus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student:

Karel Bondan Andoro Herdito

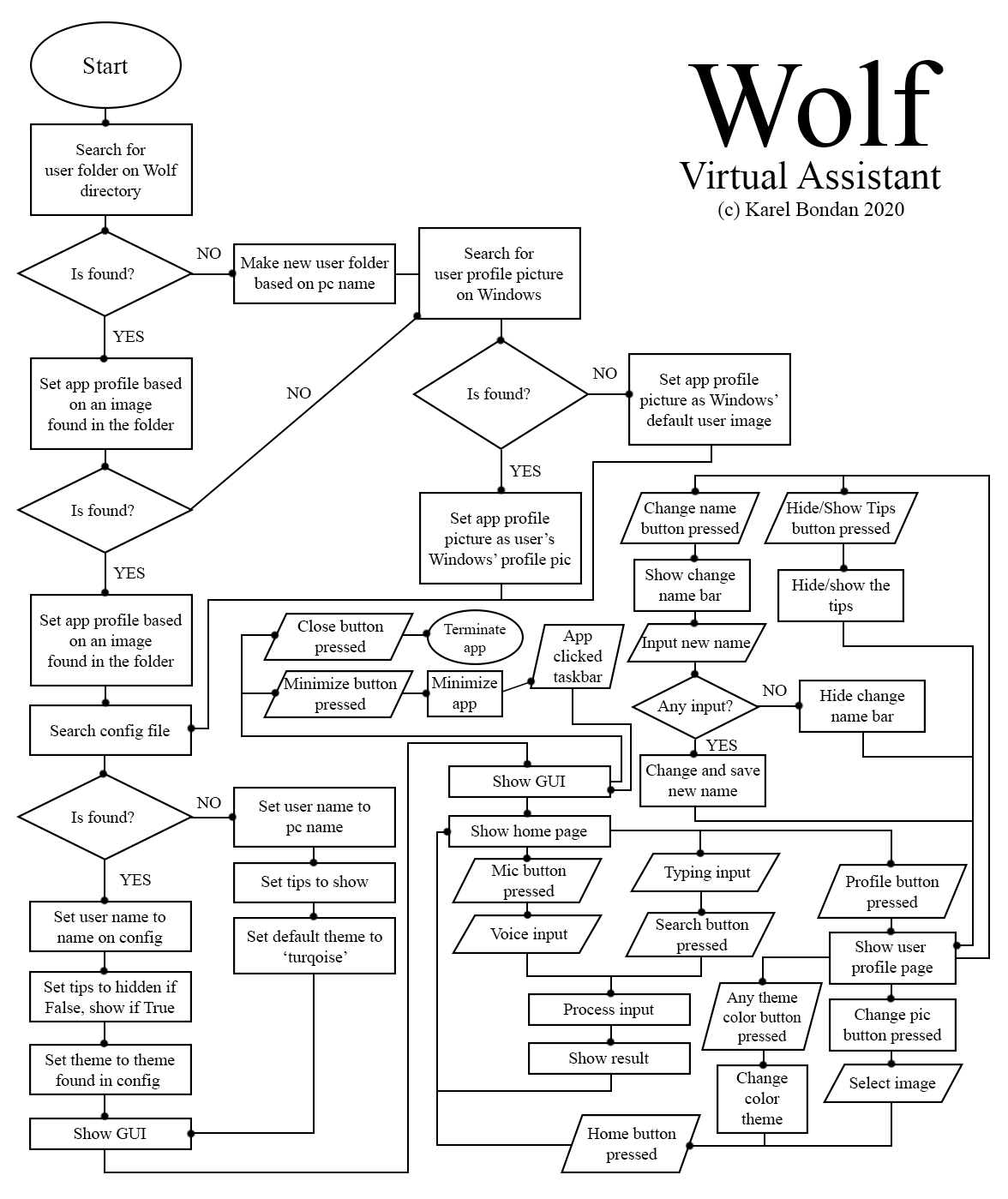
**“Wolf – Virtual Assistant”**

**Name : Karel Bondan Andoro Herdito**

**ID : 2440032373**

1. **Description**

Wolf is a virtual assistant program made for Windows, specifically Windows 10. Wolf may work on other versions of Windows, but I have not tested it myself yet. Wolf features a graphical user interface, so it is friendlier to the user. Wolf is built on top of Python version 3.8.2, with the main module PySide2 for the user interface. This program is made to fulfill the need of virtual assistant on Windows, because Cortana, the pre-built assistant on Windows platform is not supported yet in most countries.

1. **Flowchart**
2. **Libraries**
3. gTTS

The main library used to output the assistant’s voice using Google’s text to speech feature on Google Translate.

1. SpeechRecognition

The library used to recognize voice input from user.

1. playsound

The library to play sound files without having to open them using 3rd party apps.

1. PySide2

The library used to make the graphical user interface. This library is the counterpart of its parent, PyQt5, developed by the same company. The advantage of using PySide2 instead of PyQt5 is the generate python script in the designer app which is used to design the app’s GUI.

1. tkinter

To open a dialog box since PySide2 doesn’t have this function.

1. opencv

This library is used to process images, resizing them based on needs.

1. Pillow

Another library used to process images, corresponds to the tkinter library to choose images.

1. requests

This library is used to make a request html output of a webpage.

1. beautifulsoup4

The library used to scrap webpages to get informations from it, such as explanations of stuff.

1. re

This library is used to find matching string(s) in a string variable. This library is the most important one beside ‘beautifulsoup4’ and ‘request’ to get an information of something from the web. (request > beautifulsoup4 > regex)

1. wolframalpha

The library used as the calculator module.

1. pygetwindow

The library used to get a currently running process on Windows by its process name. can be used to control that process such as focusing it (making it the current application that is being interacted by the user), maximizing, and minimizing the process if it has a GUI.

1. Pyautogui

The library used to get the position of image on the screen. This library is used to locate the microphone button when the user calls Wolf using its wake-up call and to screenshot the screen.

1. threading, subprocess, signal, and atexit

The ‘threading’ library is used to play the ‘ding!’ sound whenever the user pressed the microphone button. It is so that the listening function and the ‘ding!’ sound can be executed simultaneously. The ‘subprocess’ library is used to open the always-listening function of the program, so user can call Wolf whenever they want. As its name stated, the ‘atexit’ library is used a moment before the main program fully stopped. This library is used to trigger the break (CTRL^C) signal to the subprocess program (the always-listening function) that uses the ‘signal’ library. After it’s sent the signal, the subprocess program will be terminated. ‘atexit*’* is also used to trigger the save user configuration method to save user’s current color theme, name, and the show/hide tip bool.

1. os and shutil

Both of these libraries are used as the file management libaries. ‘os’ is mainly used to make a new directory and renaming files while ‘shutil’ is used to copy and paste files.

1. socket

This library is used to get the user’s pc name to set the initial user name to their pc name.

1. time

Used to dispay the current time.

1. webbrowser

To open a link in a new tab of the pc’s default browser.

1. ctypes

To tell Windows that the program is a different process from its main program, Python, so that a custom icon which will be shown in the taskbar can be set.

1. random

To randomize Wolf’s phrase after it’s successfully shown a search result to the user.

1. Openweather and jokes API

As each name state, openweather is the API to get information of a weather of a city, and jokes API is the API to get jokes.

1. **Graphical User Interface**

* **Home page**

The home page has some parts, they are:

1. The Command Bar

The Command Bar is the longest bar on the bottom of the app. There, user can type commands so Wolf can do what user want. It is connected to the Search Button.

1. The Search Button

The Search Button is the typing command button. If user're not in the mood to say a word, user can use this feature instead. Wolf won't say anything if user use this command.

1. The Microphone Button

The Microphone Button is the voice command. User'll be prompted to say a command and Wolf will do that command for user. Wolf will also say things, different from the type command.

1. The User Profile Button

This button is where user has the freedom to customize the GUI.

1. The Home Button

To go back to the search page so user can ask Wolf other things.

* **User profile page**

The user profile page has some customizations:

1. Change user name

Wolf will immediately know that user has changed their name. The next time user asks Wolf their name, Wolf will recognize them as their new name.

1. Change user profile picture

User can change their profile picture by clicking on the big circle above their name. The supported image file extensions are JPEG, JPG, PNG, and BMP.

1. Change the color scheme

19 color schemes are available on the user profile page. User is free to choose whatever color scheme they like.

1. Show/hide the tips

By default, the tips are shown to user. If user is satisfied enough with the tips, they can hide them by clicking the ‘Hide Tips’ button on the top left corner of the page. To show the tips again, user can press the ‘Show Tips’ button on the exact same location as the ‘Hide Tips’ button. Depending on user’s choice, Wolf will hide/show the tips on next launch.

1. **Functions**
2. **Calling Wolf**

With its background listening functionality, user can always call Wolf whenever they want. The wake up calls are:

*>>> 'woof''*

*>>> 'wolf''*

*>>> 'hey wolf'*

*>>> 'woof woof'*

*>>> 'wolf wolf'*

When user have successfully called Wolf, a short 'ding!' will be heard indicating that Wolf is ready to accept commands from them. It may take some time before Wolf finally response. Give 1 or 2 seconds. If Wolf doesn't respond, user may try calling it again.

1. **Commands**

* Greets the user

Wolf can greet the user, and tell user what it can do. User can use this function by telling Wolf the following commands:

*>>> 'what can you do?'*

*>>> 'what is your name?'*

*>>> 'how do you call me?'*

*>>> 'how are you today'*

* Define something

Wolf can define the meaning of a word or telling user who a person is. The commands are:

*>>> 'define (word)'*

*>>> 'what is (word)'*

*>>> 'who is (person)'*

After Wolf has finished talking, the phrase it just said will be displayed on the main app. The thing to remember is that sometimes, Wolf can say inaudible words. It means that Wolf is telling something in user region's language. For instance, sometimes Wolf will tell user the description of someone in Indonesian if user is currently in Indonesia.

* Web searches

Wolf can automate web searches. Some search engines is implemented in Wolf so that user can use userr favorite search engine to do a search. Currently, the supported search engines are:

- Google

- Bing

- Yahoo

- Duck Duck Go

- Reddit

- Stack Overflow

- YouTube

- YouTube Music

By typing, user can type:

*>>> 'search "(user’s search query)" on (search engine)'*

Important note is that the double quotation marks must bepresent just like when user command Wolf to make a new note by typing. The example is:

*>>> 'search "cats and dogs" on yahoo'*

Wolf will take user to the Yahoo search page with userr search query already entered automatically. By voice command, user can say:

*>>> 'search on (search engine)'*

It will take some time before Wolf will prompt user for a search query. A 'ding!' sound can be heard after Wolf has finished talking. It is the indicator that it is ready to listen to user’s search query. In this state, Wolf will always listen if it fails to get the input. If user wants to cancel this process, they say anything to Wolf.

For YouTube and YouTube Music searches, user can also tell Wolf to directly play a video or a music. By typing, user can type:

*>>> 'play on youtube/yt "(what video do user want Wolf to play?)"'*

*>>> 'search "(video)" and play it on youtube/yt’*

As you can tell, user have to include the double quotation marks. Here is the example:

*>>> 'play on yt "dream vs 4 hunters finale rematch"'*

*>>> 'search "dream cheated 1.16 speedrun" and play on yt'*

The same command goes to YouTube Music. Just add music after the word ‘music’ to it. Here is the example:

*>>> 'play "and so we fall the stupendium" on yt music'*

*>>> 'search and play on youttube music "won't stop rocking"'*

By voice command, the same process goes like above but user can add an 'play' to directly play a music on YouTube Music or a video on YouTube. Here is the example:

*>>> 'search and play on youtube'*

*>>> 'play on youtube'*

*>>> 'search and play on youtube music'*

*>>> 'play on yt music'*

After some time, Wolf will prompt user the search query, just say

anything user want and Wolf will play a video based on userr search

query.

* Telling jokes

Wolf can also tell jokes to the user. User can use the following phrase to make Wolf say a joke to them:

*>>> 'tell me a joke'*

* Time

Wolf can tell user the current day and time. User can use this function by typing, or saying:

*>>> 'what time is it?'*

*>>> 'what day is it?'*

* Weather

Wolf can tell user the current weather of the city user asked it the info about. For instance, user can say this phrase to display the current weather of Semarang, Indonesia:

*>>> 'what is the current weather of semarang?'*

or

*>>> 'current weather of semarang'*

It will take some time for Wolf to gather the info then tell the user the description of the current weather of Semarang, Indonesia. After Wolf has finished talking, a graphical image can text can be seen in the user interface.

* Calculation

Wolf can do basic calculation for user. The command phrase is:

*>>> 'what is (number) (operand) (number)?'*

* Open and close apps

To open apps, user can command this phrase below to Wolf:

*>>> 'open (application name)'*

or to close a currently running application, user can command Wolf using this phrase below:

*>>> 'close (application name)'*

* Make a note

Wolf have an in-built note function that can write, view, and delete note both with typing and voice command. By typing, the command is:

*>>> 'make a new note "user note"'*

the double quotes must present in order for Wolf to successfully make a new note for user. By voice command, user can say:

*>>> 'make a new note'*

Wolf will then prompt user to say what the note will be, and after a successful record of the input, Wolf will save the note into a file which can then be opened by another commands. To view a note, user can give this command to Wolf:

*>>> 'view my note' or 'open my note'*

And to delete the note, user can type or say:

*>>> 'delete my note'*

* Screenshot

The phrase for Wolf to understand that user is trying to screenshot the screen are:

*>>> 'take a screenshot'*

*>>> 'screenshot my screen'*

*>>> 'screenshot the screen'*

* System calls

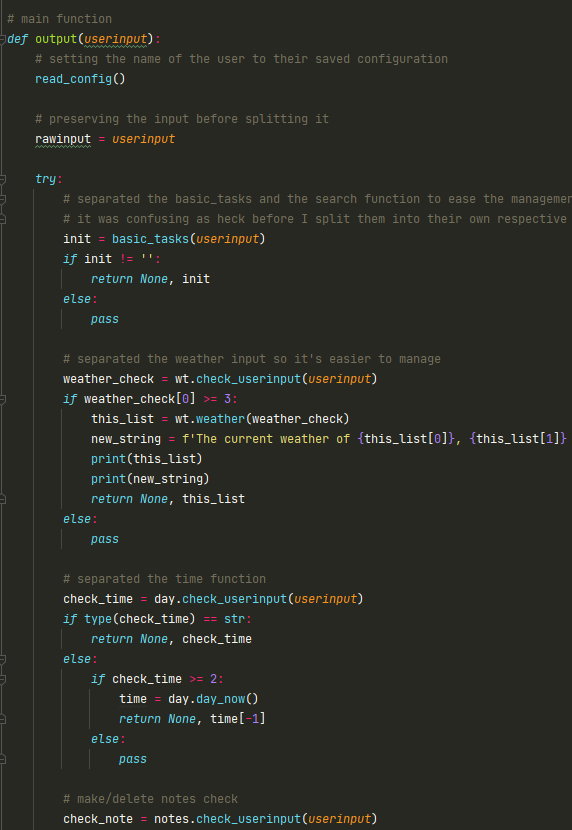
User can shut down and restart their computer through Wolf. The phrase for this to be possible is:

*>>> ‘please restart my pc’* to restart the pc, or

*>>> ‘shut down my pc please’* to shut down the pc.

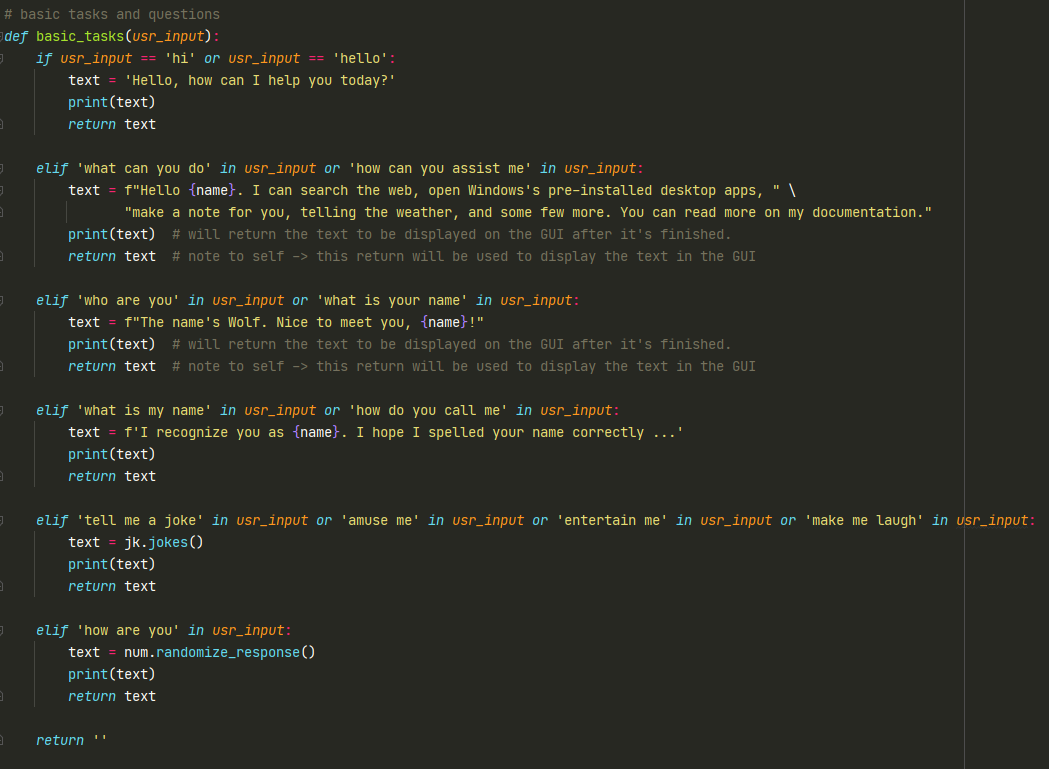
1. **Code**
2. **Introduction**

This program is made without using machine learning, thus throwing a bunch of if and else into it. The workflow of this program is that it checks the user input by splitting it. It will then check the input on every functions that has been made to predict whether the user wants to do this or not by using counter (see the examples below). If not then pass, if yes then execute that function. For example, user inputs ‘tell me the weather of jakarta now’. The program splits the input based on the blank space, then checks every single functions. If function returns value greater than or equal to 2 or 3, then it’ll execute that function. If the return value is lower than three, then it’ll pass to the next input checking.



Here, the first thing the program will check is the basic tasks input (a special occasion). If the return value is empty, then the program will go on the next one. The second one the program will check is the weather input. The weather library has the *check\_userinput*() function which will check the input.

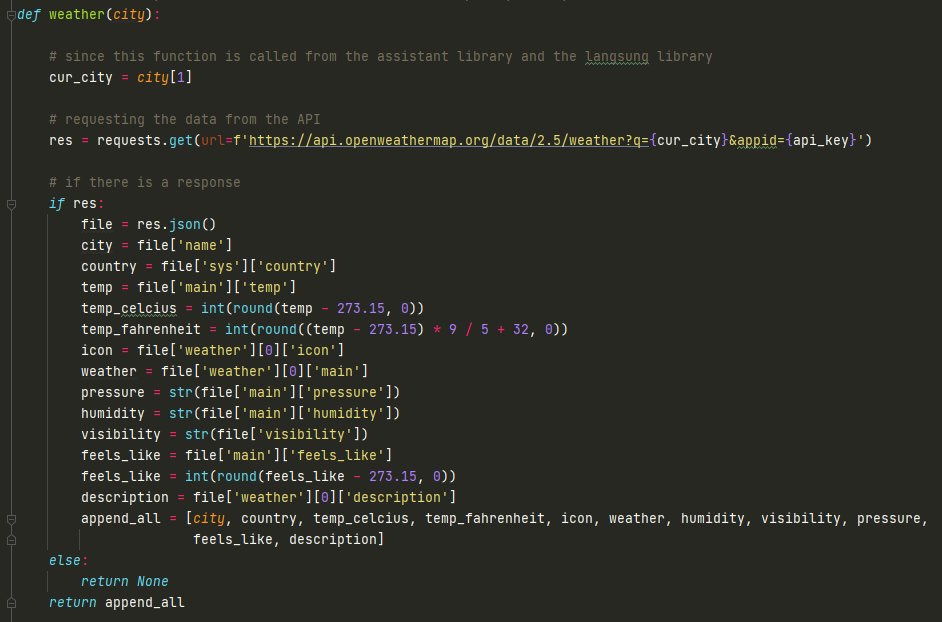


Every library that I made (*apps.py, day.py, notes.py, screenshot.py, and weather.py*)have this function to check the user input. I did not define it only once to ease the process and to avoid circular import.

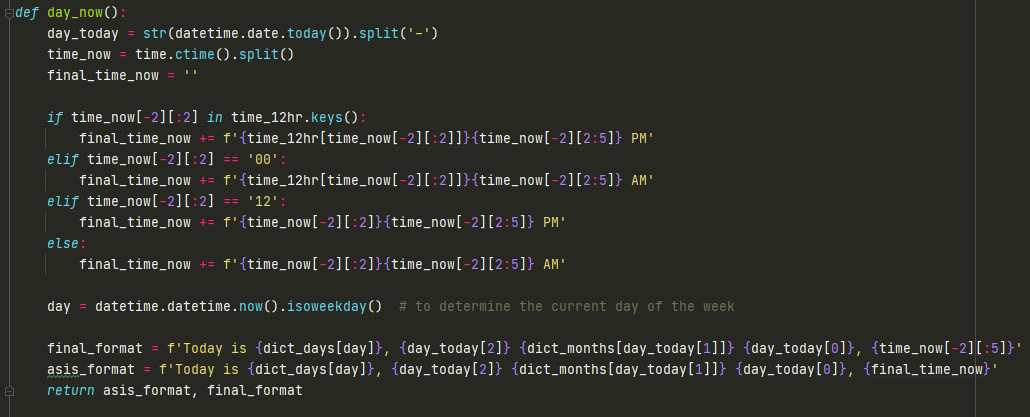
1. **Basic Tasks**

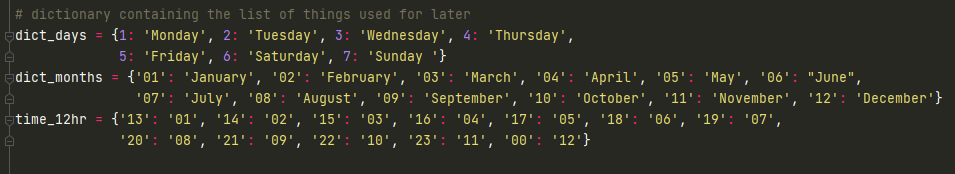
The first function is the *basic\_tasks(input)* function. As its name state, this function is executed when user commands Wolf to do simple tasks such as asking its name, what chores can it do, and so on. Below is the code for the function.

Again, because this program does not use machine learning, a bunch of *if* and *else* must be implemented for the program to work as we want.

1. **Weather**

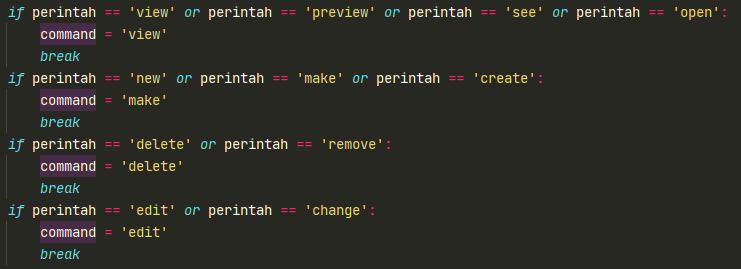
This is the weather function. It calls the openweather API to get the data. The data is received in a form of a raw string file like a json file. Therefore, it then gets converted into a json file which behaves similarly to python’s dictionary. The rest of the lines are variables to store the converted data. If the request attempt failed and received nothing, it will return *None,* else it will return a list consisting of the weather data. The main window will then pick the values from inside the list and displays it in the main app. The example of the result will be on the working app proof section of this report.

1. **Time**

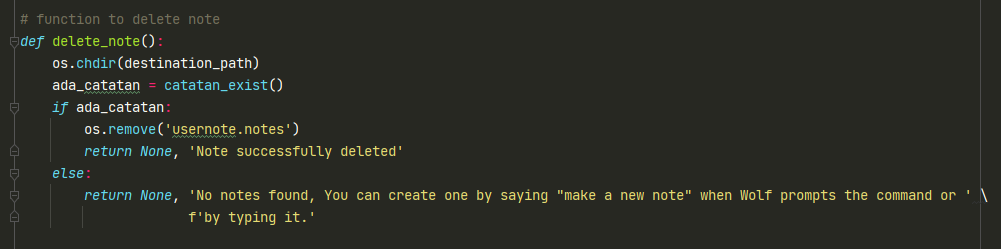
The next one is the time function. It is a simple function to display the time. This function returns 2 values, the first one is a string which will be shown in the GUI, and the second one is a string which will then be passed into the *voice\_output(output)* function which will be discussed later. To distinguish this, I made a variable that stores a dictionary containing the key, which is the current time received from the ‘time’ library, and the values which contain the 12-hour format of the clock.

As you can see, there are several other dictionaries, the first one is the ‘days’ dictionary and the second one is the ‘months’ dictionary. Since I used *isoweekday* method of the ‘time’ library, it returns a single digit based on the current day of the week. 1 is Monday, 2 is Tuesday, and so on. Therefore, I declared a new ‘days’ dictionary. The return value of the ‘time’ library for month is in the form of digit, not the month’s name itself. This is why I declared a new ‘month’ dictionary too.

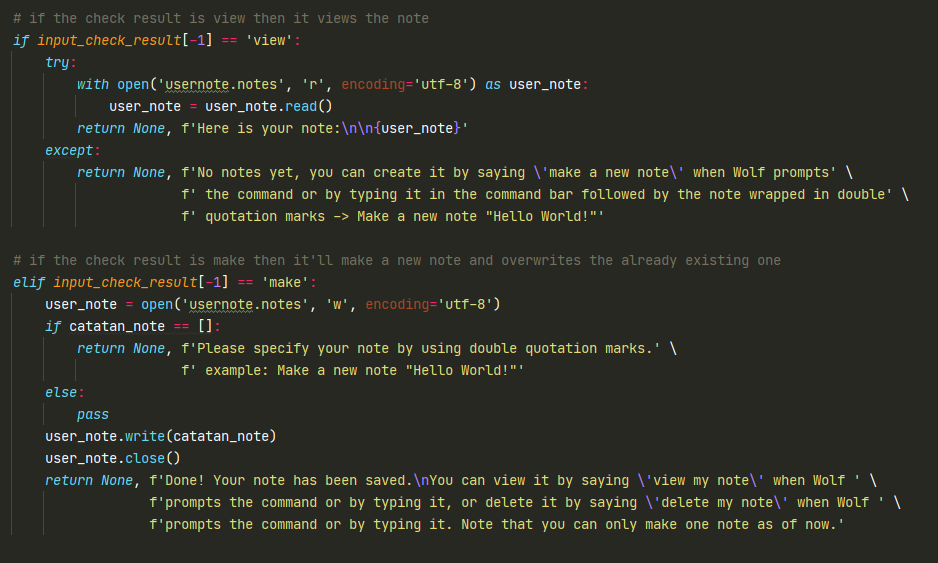
1. **Note**



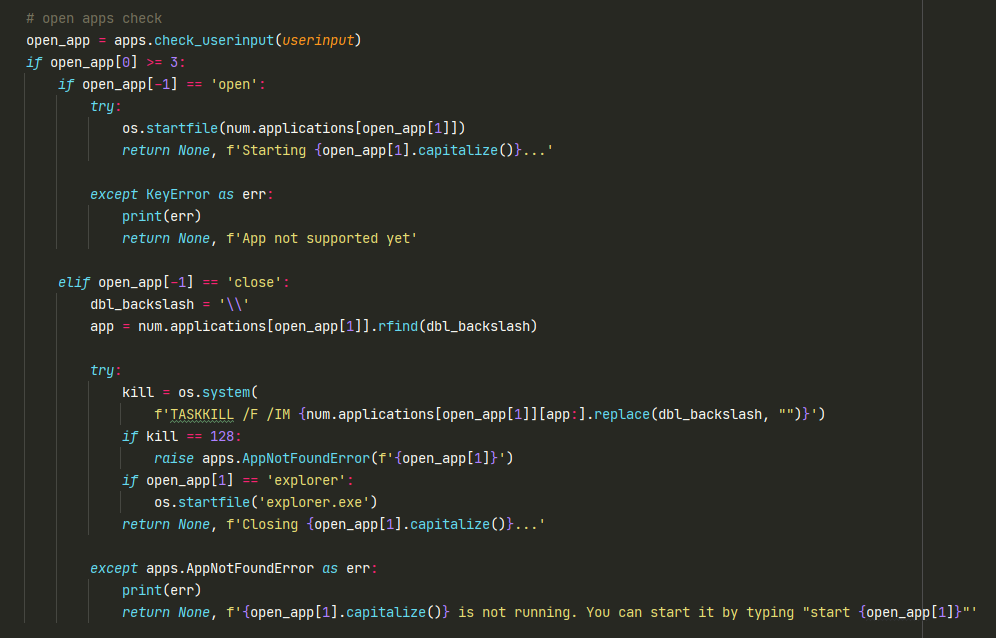
The next one is the note function. Because in one library there are multiple commands to do with the note, I use this *if* and *else* statements to determine the input. It will then return the ‘command’ variable which will determine what the function will do (make, view or delete a note).



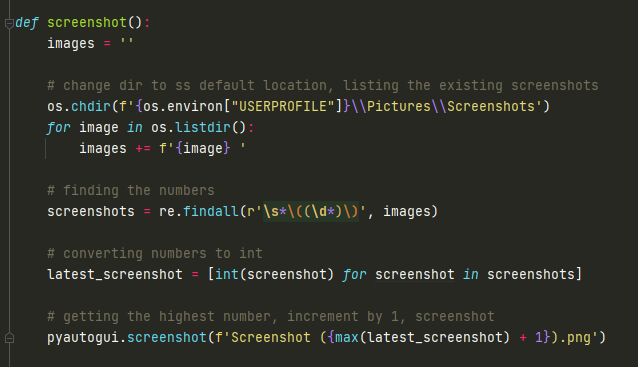
Above screenshot is the delete note function



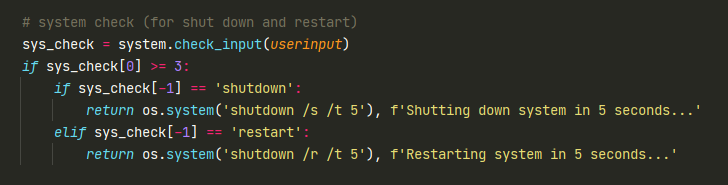
And above screenshot is the function containing the *view* and *make* feature of the note. *View* uses the ‘r’ attribute of open because the program only needs to read the note. *Make* uses the ‘w’ attribute of open because the program needs to write a new note. ‘w’ is the most efficient attribute because it can create a new file based on the given name if the file does not exist.

1. **Apps**

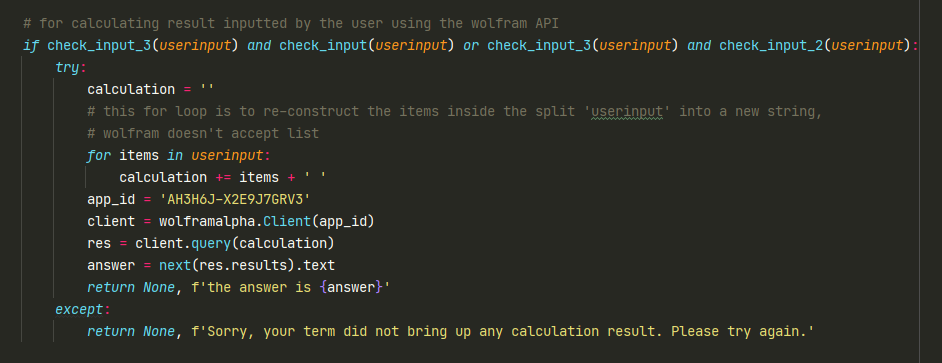
This is the open and close apps function. It uses the ‘os’ library to do the task. If user commands Wolf to open an app, it will start that app using the *os.startfile* method. Else if user commands Wolf to close an app, it will close the app using the *os.system(‘TASKKILL /F /IM’)* method. This ‘TASKKILL /F /IM’ is actually a Windows’ command, thus it can also be used on the Windows Power Shell or the Windows Command Prompt. /F is the flag to tell Windows to end the process forcefully, and /IM is the flag to get the process by name. Because closing an app is just the same as force killing an app from Task Manager, a new exception is made for File Explorer. This app won’t reopen itself when it’s killed forcefully. Desktop, taskbar, and the explorer itself will be gone if this happens. Thus, if user tries to close this app by command, Wolf will automatically start a new *explorer.exe* process which is the File Explorer app.

1. **Screenshot**

This is the screenshot function. Here, I use the ‘os’ module again to change the working direction of the program to the default Windows’ screenshot folder. Since Windows uses ascending number to store the screenshot, I had an idea about getting the highest number of the current image stored in the screenshot folder. This is why a *for* loop is declared there. The *for* loop is to loop all the files inside the default screenshot folder and appending their names into the ‘images’ variable. A regex will be used in finding the numbers. This is why I used a string variable instead of a list to store the image names, because regex only accepts string, not list. After it’s found the numbers, I converted them into integers by using list comprehension. Finally, the screenshot will be initiated and a new portable network graphics image will be created and stored inside the default folder with the highest digit in plus one.

1. **System calls**

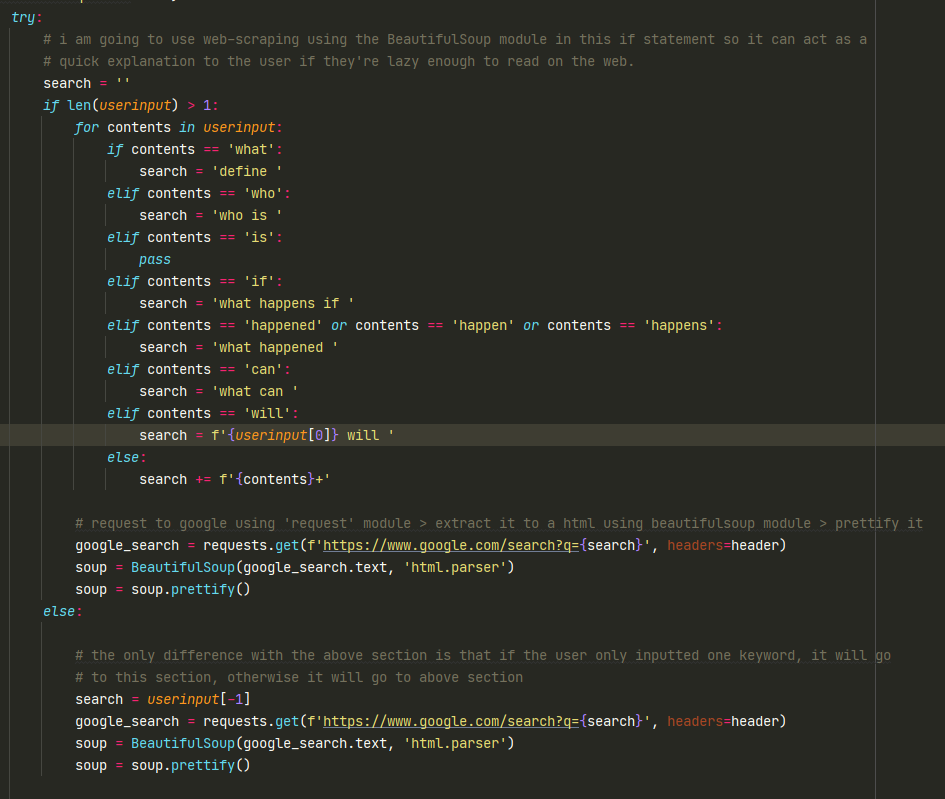
I use the ‘os’ library again to call the ‘shutdown’ and ‘restart’ command. The /s flag is short for shut down, /r is short for restart, and /t is short for time, which is the time before the system shuts down or restarts, according to user’s command.

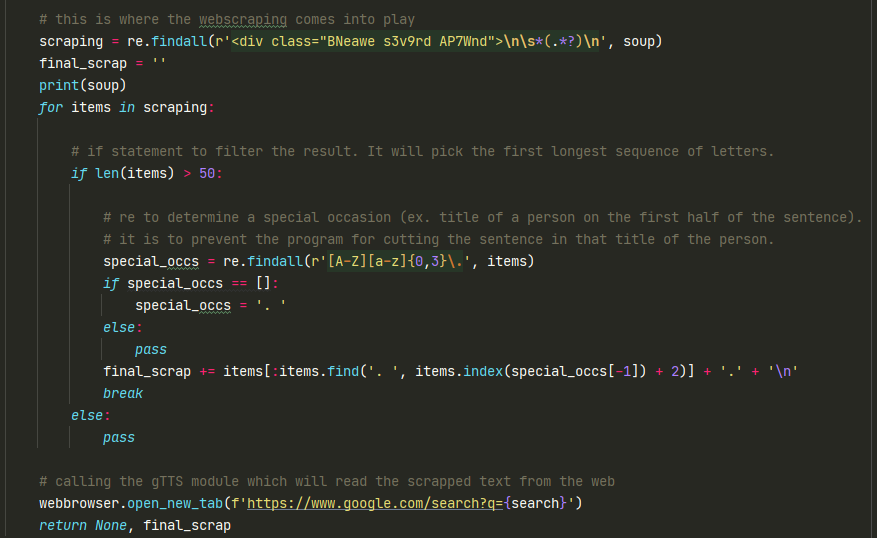
1. **Calculation**

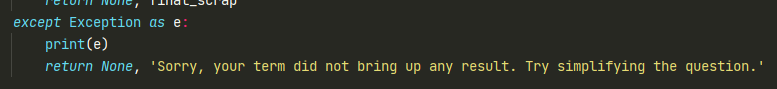
*check\_input\_3(userinput)* is a function to check operands in the user input. *check\_input\_1(userinput)* is a function to check numbers on its digit form in the user input, and *check\_input\_2(userinput)* is a function to check numbers on its word form*.* If as stated above are True, then execute the function, else pass and checks the other functions.

1. **Define**

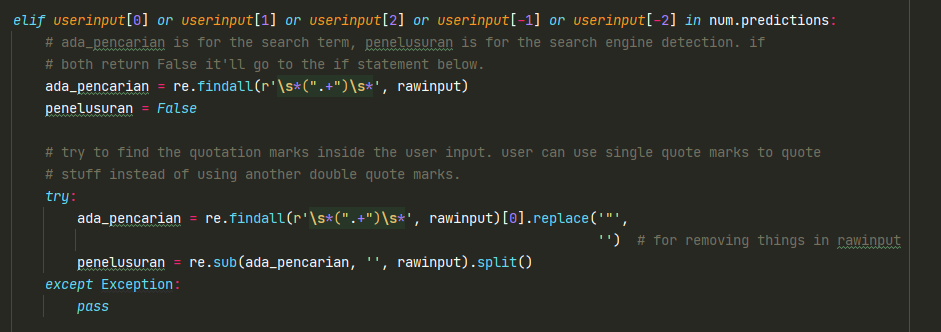
This function uses web scraping from Google to display the result into the GUI. I found out that Google can detect that its search is being automated by a script, thus it will then return a captcha which then breaks the program. This header dictionary here will solve the problem. By defining the header above, we can bypass the captcha because we are using the header based on the user’s actual browser, Chrome for example. After defining the header, I then declare a new *if* statement, as shown below:



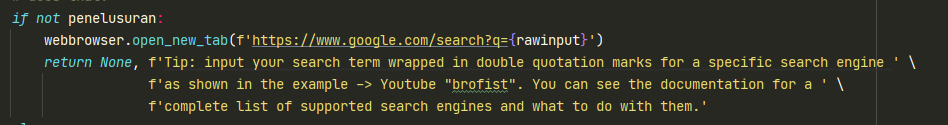
This *if* statement is declared to differentiate between one keyword input and multiple keyword input. For example, if user inputted ‘define gravity’ then it’ll go to the *if* statement, else if user only inputted ‘what’ it will go to the *else* statement. After it went to one of the statements declared, it will then request the result from Google using the ‘requests’ library, defined in the variable ‘google\_search’. After that, ‘beautifulsoup4’ will do the scraping, storing it in a variable, in this case the variable is ‘soup’.

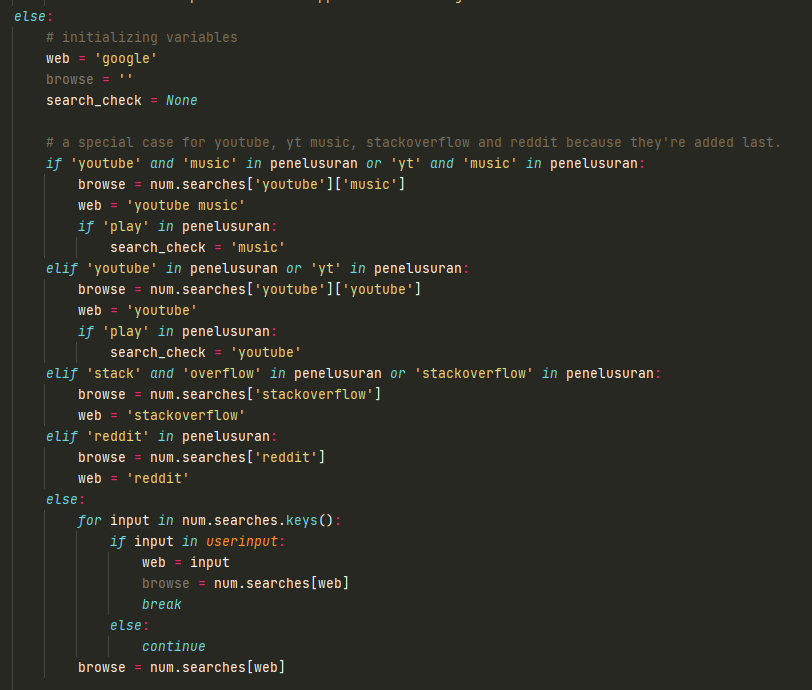
The web scraping part comes into play in above screenshot. The regex library is being used there to find the search result. Because it will find many strings, I then defined a *for* loop to loop all the regex result stored in the list, which is stored in the ‘scraping’ variable. If the length of the string is greater than 50, that string will be cut to only a sentence long to keep the aesthetics on the GUI and so Wolf isn’t too long reading alout the result. The idea is to cut the string when it finds a period followed by a blank space. But since results can show titles, I then make a new regex to exclude titles. It will be the parameter for the .*find* primitive string method. If the regex finds title(s), it will go to the *else* statement, else if it doesn’t find any title, it will then set the parameter to a period followed by a blank space. After that, the string will be cut and be restored in a new variable called ‘final\_scrap’. Finally, it will open a new tab in the pc’s default browser, and at the same time if user inputted the query using voice command, Wolf will read the result aloud. Finally, if an exception is raised during the execution, it will then go to the exception block as shown below:

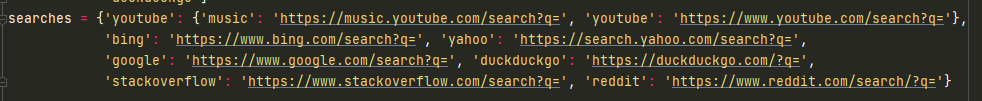
‘requests’ module can sometimes be at timeout if the response from the server is bad. This can happen if it doesn’t find any search result, thus it’ll throw an error. To prevent things from breaking, the exception block is made. The message will be shown in the GUI as well to tell the user that the search attempt failed. User may try simplifying the question; yes, most of the time when I test, complicated queries can throw this kind of error, this is why the message is to simplify the question.

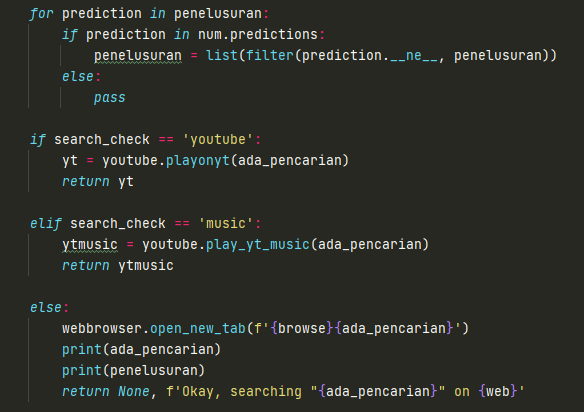
1. **Search**

The last function is the search function. *num.prediction* is a list where tons of predictions are there as the index. To give more thorough look, the list is shown in below screenshot:

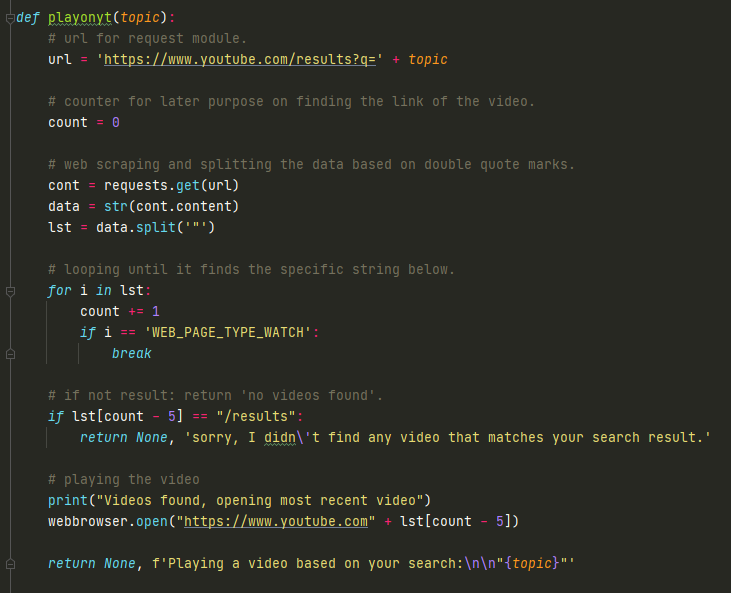
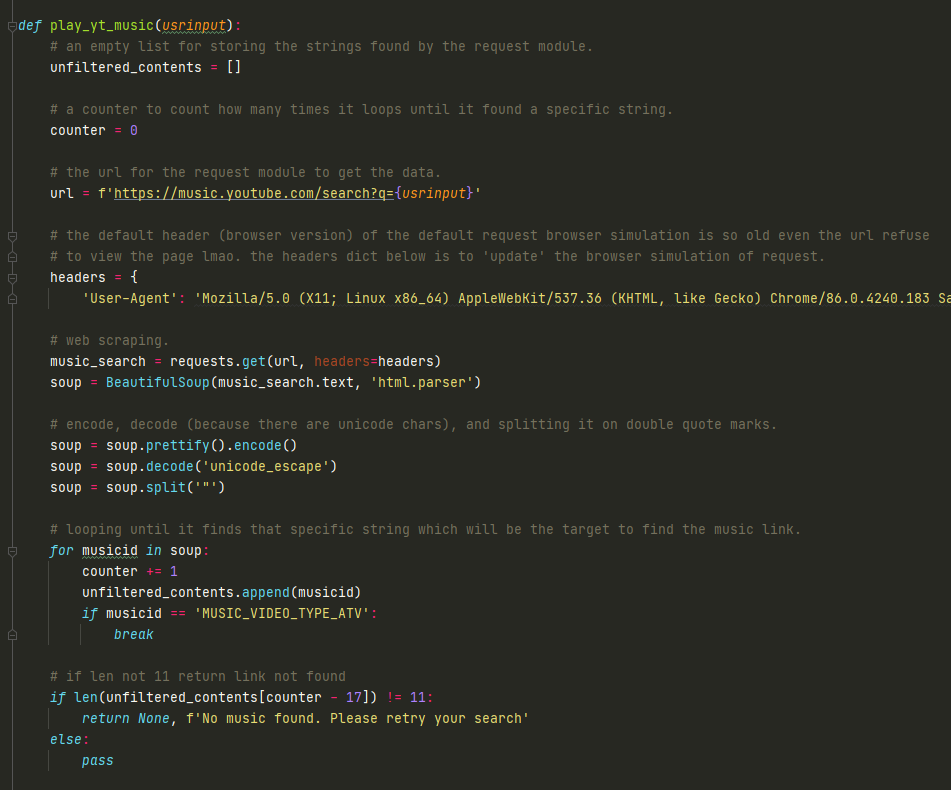
As you can see, there is a regex if the code goes to this if statement. This regex is to find the search query wrapped in double quotation marks, as explained in the documentation above. If it finds the double quotation marks, it will then make the ‘penelusuran’ variable store a value, thus automatically making it True.

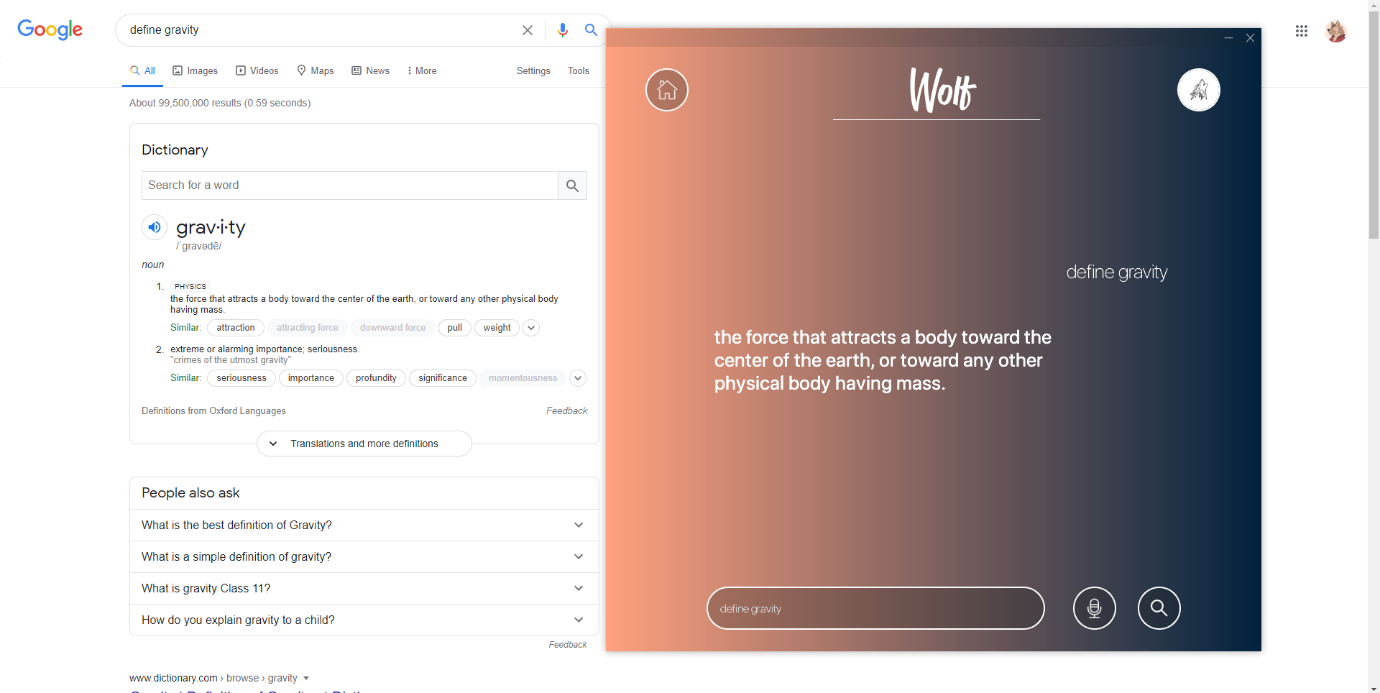
If ‘penelusuran’ is False, then it will open a new tab in the browser with the user input as the search query. Else, if a value is stored in the ‘penelusuran’ variable:

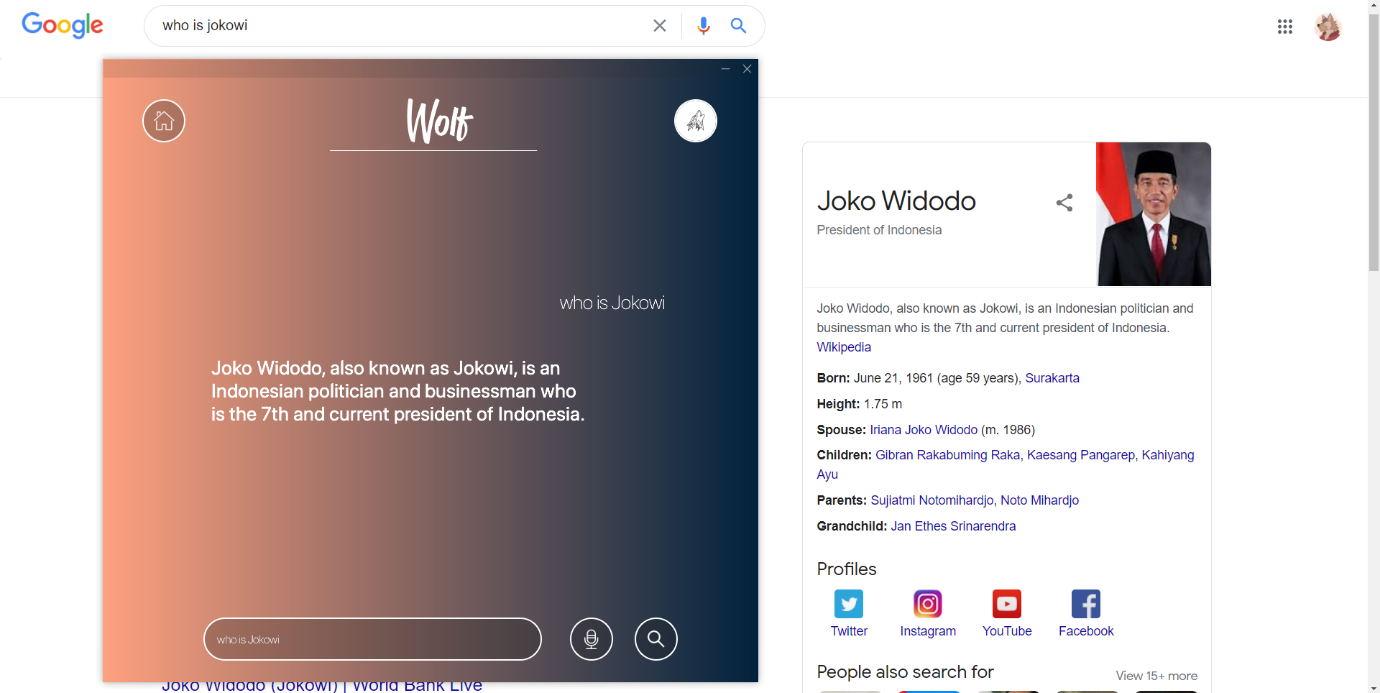
It will go to this *else* block which the main function will execute. There will be a few new variable in order for the output to work correctly. The first variable is ‘web’, which is the website that will be used to search the query. As explained in the documentation above, there are several websites that can be used as the search engine for the query. If user defined the search engine, it will set the ‘web’ value to that search engine. Else, the default will be Google and it’ll search the query on Google. A special case is done to YouTube, YouTube Music, Reddit and Stack Overflow. YouTube can be shortened to just ‘yt’, it also has its counterpart YouTube Music; Stack Overflow can be appended as one word or separated as two words; and Reddit is added last, these are why a special case is applied to said search engines. A special variable called ‘search\_check’ is declared for YouTube and YouTube Music’s case. If the program goes into one of these blocks and there is a ‘play’ word in the input, it will set the ‘search\_check’ variable according to the block it went to. If it went to the YouTube block then it’ll set the ‘search\_check’ variable’s value to ‘youtube’, else if it’s the YouTube Music block then it’ll set the ‘search\_check variable’s value to ‘music’. If the program didn’t find any matching statements, it will go to the *else* block, which will go into a *for* loop. *num.searches.keys()* contains below items:

If any of the keys are inside the user input outside the double quotation marks, it will then set the variable ‘web’ to that search engine, then set the variable ‘browse’ to that search engine’s search query link, which is the value of the said dictionary.

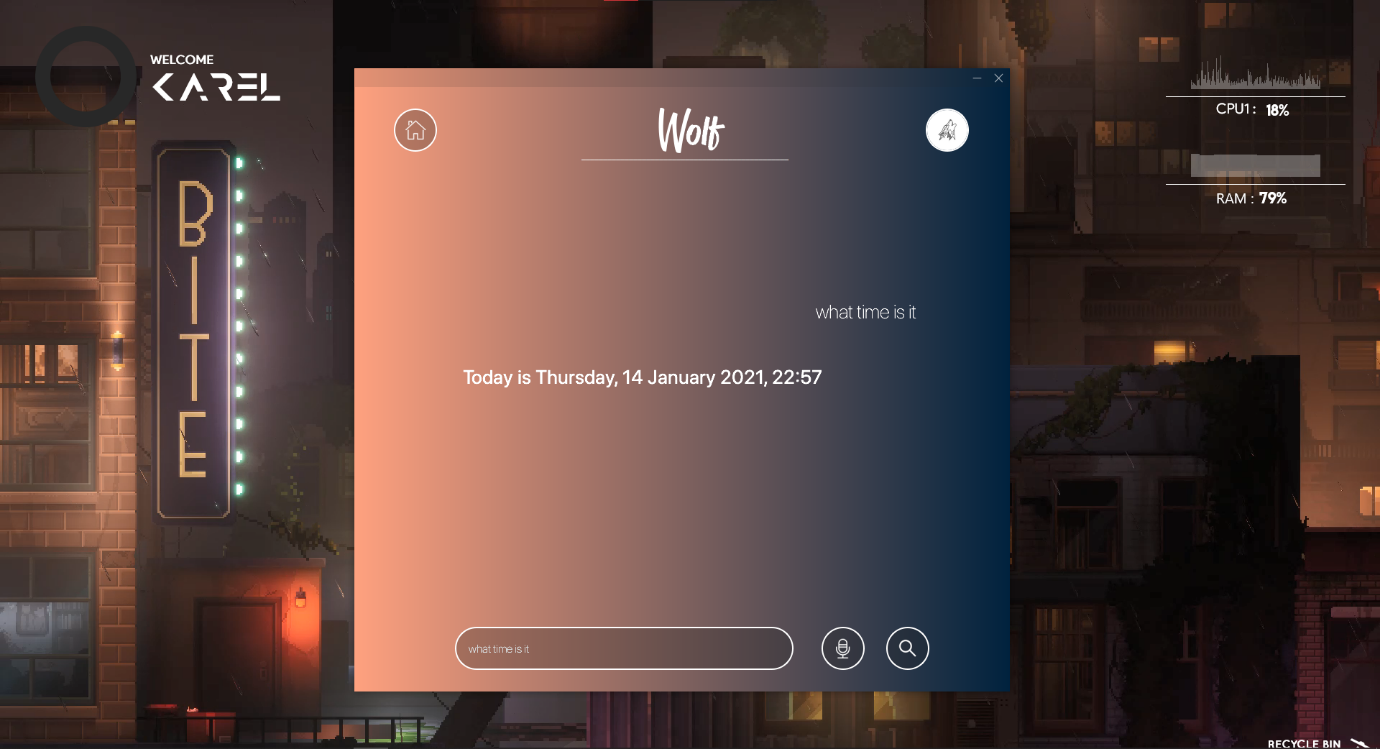
The program will then go into another *if elif else* statement. If the value of ‘search\_check’ variable is as shown in the picture, it will either play a video on YouTube or play a music on YouTube Music, depends on the user input. Else, if ‘search\_check’ is *None* then execute the lines that are in the *else* block, which is searching the query on the website the user has specified before, or on Google if the user didn’t specify any search engine. Here is the function to play a video on YouTube or play a music on YouTube Music. The first screenshot is YouTube, and the second one is YT Music:



1. **Program**

Proof of the working program when user commands Wolf to define something. In this case, user commanded Wolf to define the word ‘gravity’.

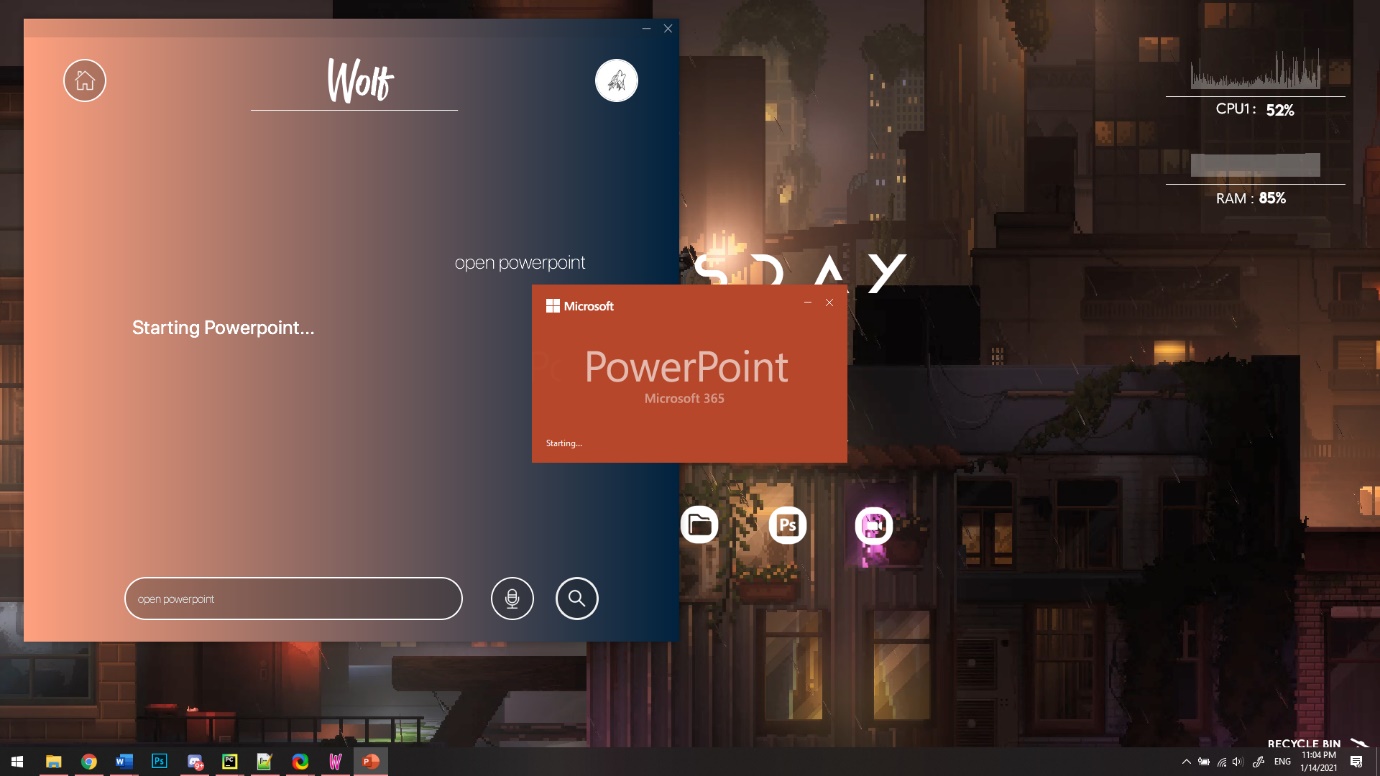
Proof of the working program when user commands Wolf to define someone. In this case, user commanded Wolf to tell him who Jokowi is.



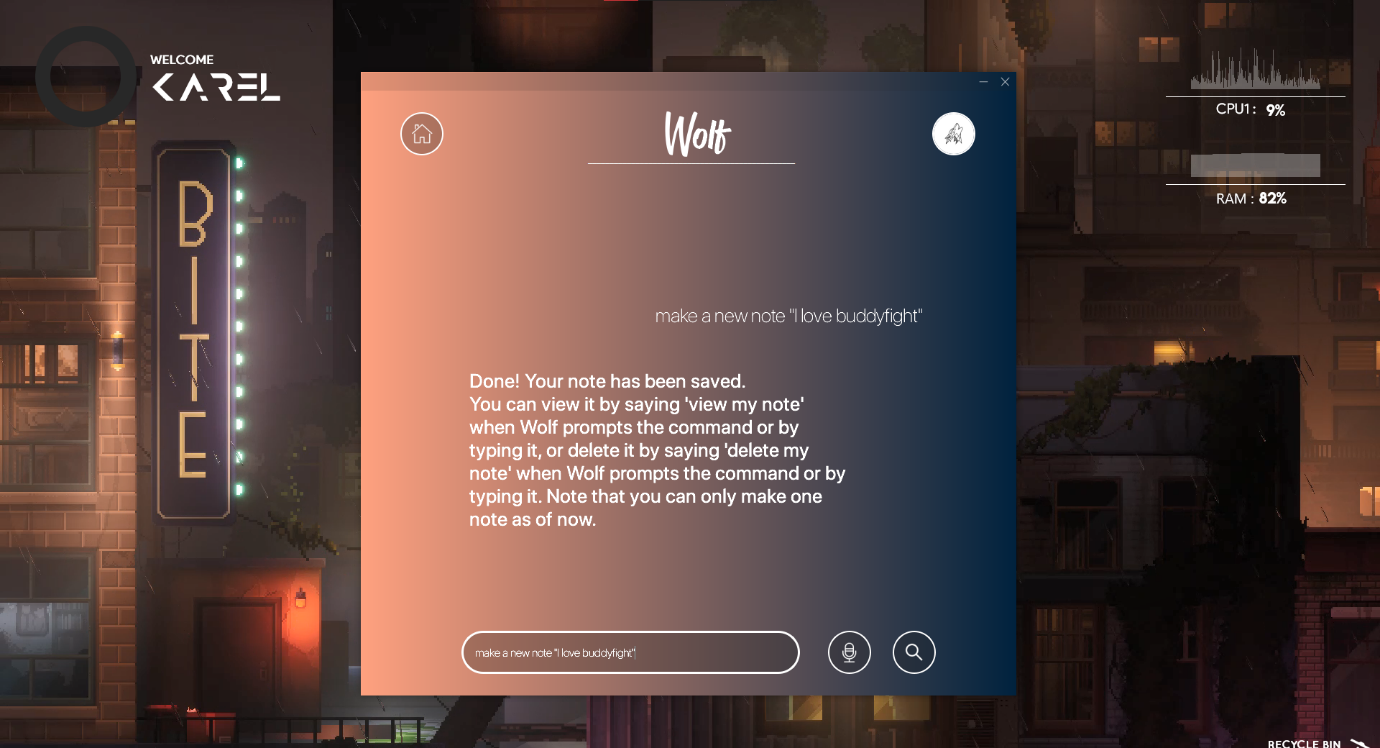
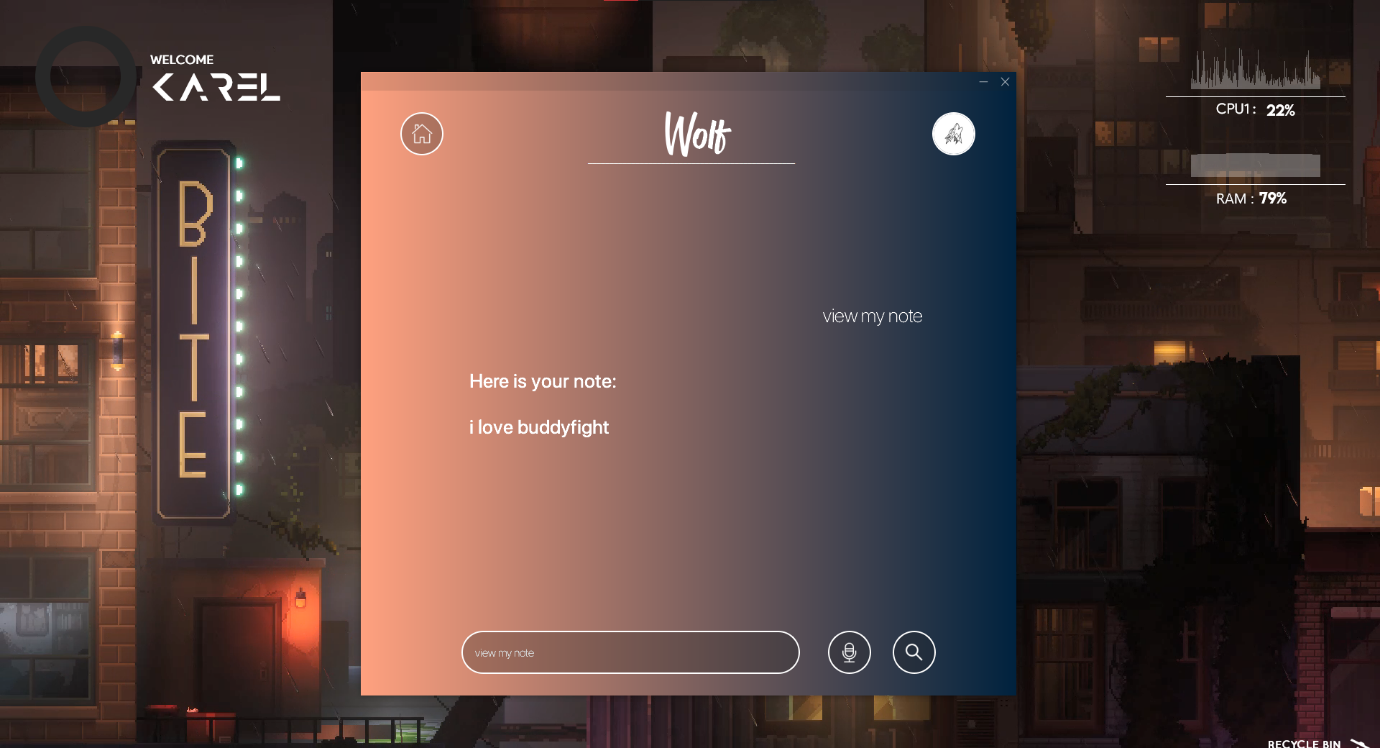
Proof of the working program when user commands Wolf to tell the current time.

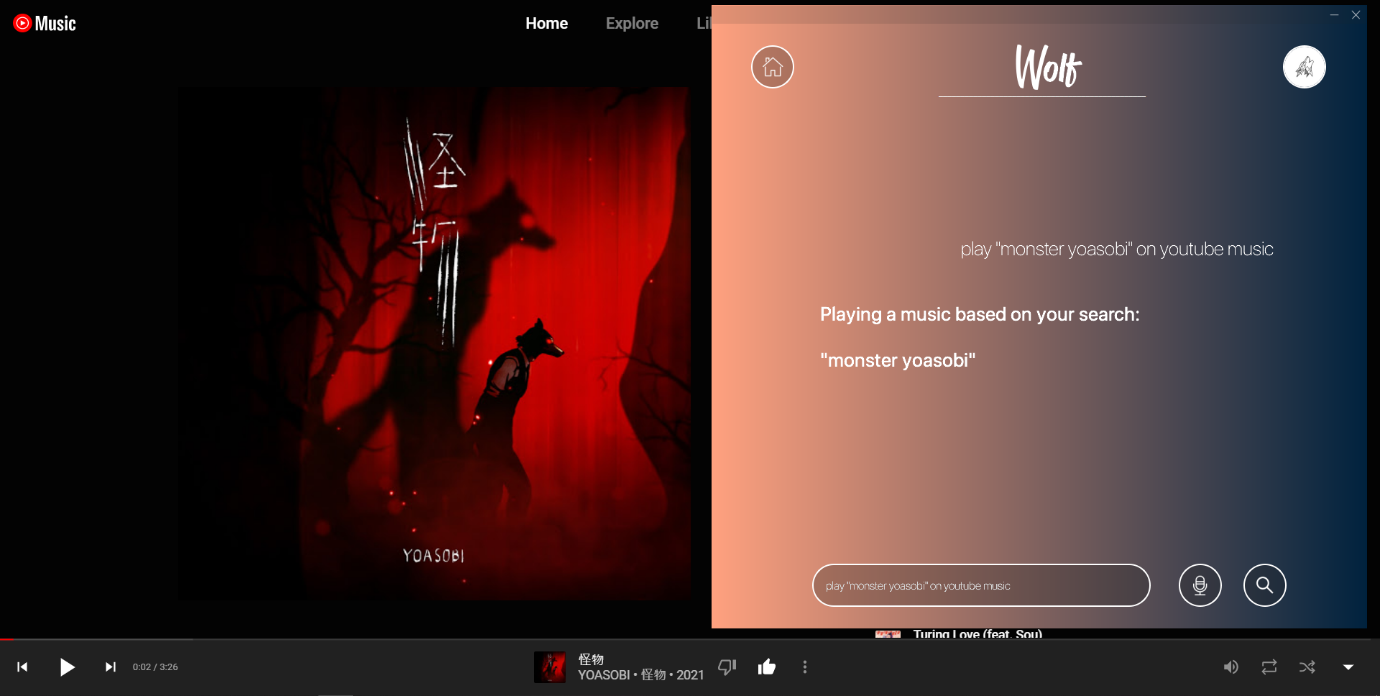


Proof of the working program when user commands Wolf to tell the current weather of Paris, France.



Proof of the working program when user commands Wolf to open an application. In this case, the application is Microsoft PowerPoint.



Proof of the working program when user commands Wolf to make a new note and viewing it afterwards.

Proof of the working program when user commands Wolf to search and play a song on YouTube Music. In this case, it is Yoasobi – Monsters.

1. **Final Words**

I learned a quite a valuable lesson while making this program. Finding new syntaxes and modules of course, of course, but the main thing that I learnt during this process is not to have high expectations of the outcome. I tried implementing an alarm function and it fails miserably; it cannot work simultaneously with the main assistant program. I’ve tried literally everything: subprocessing, multithreading, multiprocessing. Nothing works. This gives me a valuable lesson to learn, that is not to expect high outcomes of anything, even the littlest things. Finally, I thank Sir Jude that has guided me throughout this semester. I learn a lot of new things in the programming world and can finally experience coding thoroughly. Thank you, sir. My profound apology, if ever I have done something that was, or were not pleasing to you. Shall we meet in the next semester, I would be more than grateful to be your student again.

1. **Project**

The link for this project: <https://github.com/karelbondan/Wolf>