**Odd Semester (2019/2020)**



**BINUS UNIVERSITY**



**BINUS INTERNATIONAL**



**Assignment Cover Letter**

**(Teamwork)**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | | |  |  |  |  |
| **Student Information**: | **First Name** | | |  |  | **Last Name**  **Rasendriya Rasyid**  **Sianandar**  **Hisyam**  **Michael Halim** | **Student ID Number**  **2201798295**  **2201796440**  **2201797430**  **2201798761** |
| 1.  2.  3.  4. | **Alifio**  **Jason**  **Muchsin**  **Nicholas** | |
|  |  |  |  |  |  |  |  |
| **Course Code** | **: COMP6340** |  |  |  |  | **Course Name** | **: Analysis of Algorithms** |
| **Class** | **: L3AC** |  |  |  |  | **Name of Lecturer(s)** | **:** **Maria Seraphina** |
|  |  |  |  |  |  |  |  |
| **Major** | **: CS** |  |  |  |  |  |  |
| **Title of Assignment**  (if any) | : **Voice Recognition** |  |  |  |  |  |  |
| **Type of Assignment**    **Submission Pattern** | **: Final Project** | |  |  |  |  |  |
| **Due Date** | **: 29 - 10 - 2019** | |  |  |  | **Submission Date** | **: 29 – 10 -2019** |

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:

1. Alifio Rasendriya Rasyid 2. Jason Sianandar 3. Muchsin Hisyam 4. Nicholas Michael Halim

**CONTENTS**

**1. Introduction 4  
 1.1. Background 4  
 1.2. Problem Description 4**

**2. Related Work 4**

**3. Proposed Alternative Solution 5**

**4. Implementation 5  
 4.1. Libraries Used 5  
 4.2. Implementation of Parsing 6   
 4.3. Step-by-step example 8**

**5. Program Manual 9**

**6. Discussion 10  
6.1. Conclusion 10  
6.2. Recommendation 11  
 6.2.1. Offline Capabilities 11  
 6.2.2. Machine Learning 11**

**7. References 11**

# Introduction

Our final project will revolve around using a speech-to-text algorithm combined with our own parsing method to create a voice command program. This is a compiler-oriented topic. We aim to create a program which helps the user do their tasks on a Windows computer by adding quick voice convenience-oriented command controls while the program is running where the user can simply push a hotkey and speak.

**1.1 Background**

With new technologies growing rapidly in this digital age, mankind must adapt to new situations all the time but not all can adapt to the rapidly growing technology. People who are old or people who were not exposed to new technologies might have difficulties to adapt. On the other hand, power users may want to do several commands at once without touching their computer. That’s why we thought of the idea on how we can incorporate speech recognition program into our everyday life to ease those difficulties. Those who are not “tech savvy” wouldn’t need to learn the functions and the complexities of the new technologies, while power users can chain commands to perform tasks quickly.

## **1.2 Problem Description**

We needed a way to speak what we want to do to the computer and lets us do multiple tasks at once. While current voice assistant allows us to do many things, we saw a problem where activating the voice assistant itself can be a hassle. We wanted to make something that allows for quick and simple spoken commands that can be chained for efficiencies.

# Related Work

There are several related applications to ours. Google Assistant, Siri, Alexa. While those other programs have enough data to do many different things with the help of context recognition, we aim our program to be simpler but can do many things at once. This is because we implement our own method of parsing the commands and made them as simple as possible to understand, speak, and chain.

## **Proposed Alternative Solution**

From the problem discussion, we would like to make a Python application, based on speech recognition that can help the user to do their tasks on a Windows computer by adding quick voice convenience-oriented command controls while the program is running. We chose to use an existing speech recognition library in python called SpeechRecognition. This library also uses the help of Google’s Speech Recognition algorithm.

We will then implement our own method of natural language processing (NLP) by parsing the commands given by the user based on key command-words and key object-words. Because of a simpler set of commands, we may be able to parse the commands using a top-down approach.

1. **Implementation**

**4.1. Libraries Used**

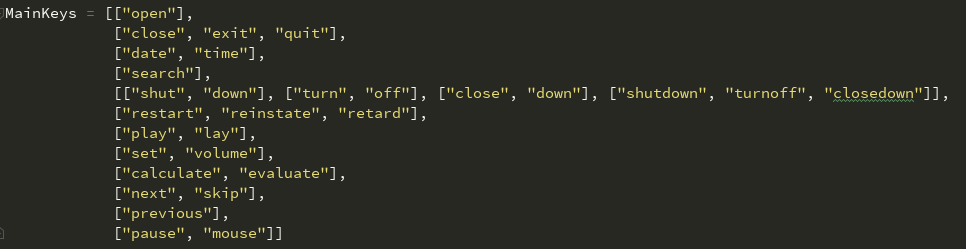
* **time, time access and conversion:** This module provides time - related functions.
* **os, Miscellaneous operating system interfaces:** This module provides a portable way of using operating system dependent functionality.
* **re, Regular expression operations:** This module provides regular expression matching operations similar to those found in Perl.
* **Keyboard, Full control of the keyboard:** This module enables you to hook global events, register hotkeys, simulate key presses.
* **win10toast, Notifications:** This python library enables you to display Windows 10 Toast Notifications.
* **speech\_recognition, Speech Recognition:** This python library enables you to perform speech recognition, with support for several engines and APIs, online and offline.
* **googleapiclient.discovery :** A client library for Google’s discovery based APIs.
* **datetime:** this module supplies classes for manipulating dates and time in both simple and complex ways.
* **ctypes:** A foreign function library for python. It provides C compatible data types, and allows calling functions in DLLs or shared libraries.
* **PyQt5:** [PyQt](https://riverbankcomputing.com/software/pyqt/intro) is a library that lets you use the [Qt GUI framework](https://www.qt.io/) from Python. Qt itself is written in C++. By using it from Python, you can build applications much more quickly while not sacrificing much of the speed of C++.
* **threading**: Enables multiple processes at once.
* **QThread:** A part of PyQt. Threading for PyQt5.

# 4.2. Implementation of Parsing

Our project focuses on how we can parse a person’s speech or written sentence into a command that can be executed by python. The approach that we chose is by scanning the text for keywords and objects or nouns that can be operated on. This approach is inspired by bottom-up parsing.

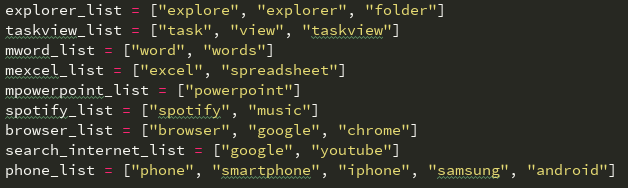
First, our program splits the input by the user (either it’s from speech to text, or directly typing it on the input box) by looking for the words “then” and “and.” This first step is crucial in chaining the commands and avoid the program from activating the wrong command from a long sentence.

Second, our program looks for important keywords that signals it to look for another set of object keywords. The following picture shows our current list of keywords. Note that we designed this program with modularity in mind, thus more keywords can be added in the future without much hassle with the rest of the code.



These MainKeys allow us to guess which object we would need to operate on. For example, the word “open” would prompt the program to start looking for application names next. Another important thing in determining the next set of object keywords is in the priority of the MainKeys checking. For example, keywords “search” and “play” have the same highest-priority to be checked. This is because we want the program to be able to search, or play songs of different titles, with as many words as possible. We don’t want to say “search how to open explorer on google” and end up opening windows file explorer instead of searching “how to open explorer” on google.

MainKeys are placed in a two dimensional array to allow for modularity and addition of new words in the future without messing with the code too much. The next step is to look at which objects can be operated on by the previous keyword. These objects are placed in the following lists:



These lists allow us to add error adjustments to the list of accepted words. For example, a usermay say “open explore” instead of the correct command of “open explorer.” This would still prompt the correct command because we added the possibility of the user saying different “variants” of the words.

If we refer to how Context Free Grammar is related to how our parsing works, it can be said that our parsing can be defined in the following way:

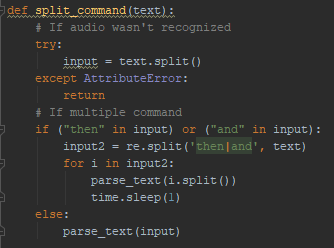
https://lh3.googleusercontent.com/hWhgjvPK5bwnhNFwTobUox3_Otkh5tpN2cddtOugP-0-hqY4WFmHDzK8Kn2hDSQnJCQ-39LzDSyQNy1VgVFndPcojbAxKMMA-xuObVtYHrxNeML3XTZMQPLU4z3LBEg56pXm9X92

* V = Words in the MainKeys and some of the lists from the lists of objects
* ⅀ = some of the set of words in lists of objects
* R = The relations can be found deep in the functions such as parse\_text and open\_stuff. These functions act as checkers to see whether the MainKeys are paired with the correct object words or to process the input from the user accordingly so that different patterns of speech can still work.
* S = The start symbol can be found in MainKeys.

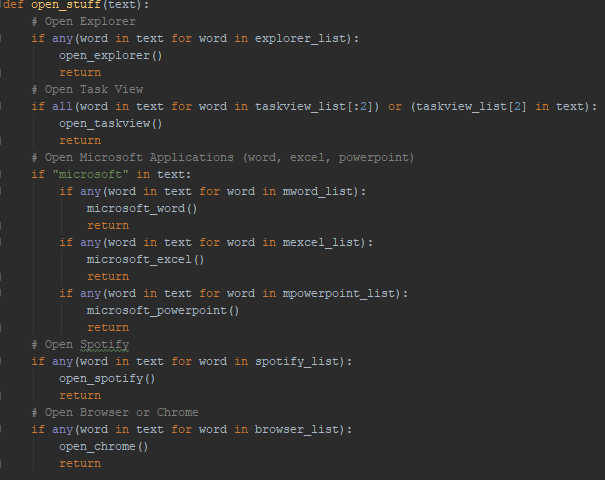
# 4.3. Step-by-Step Example

Command: “Computer open Spotify and then play something on Spotify”

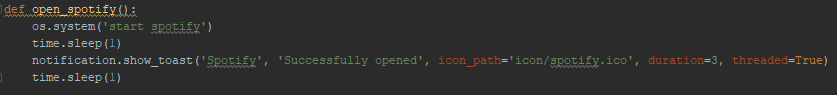
1. First, the split\_command() separates the sentence into three parts:
   1. “computer open Spotify
   2. “” an empty string due to words “and” and “then” used together
   3. “play something on Spotify”



1. Following sub-sentence a, the parse\_text() function detects the word “open” as a main key and invokes open\_stuff().



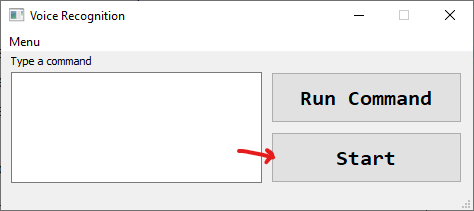
1. Sentence a then gets scanned for object words and the word “spotify” is found. Thus, the open\_spotify() command is called to open the application.



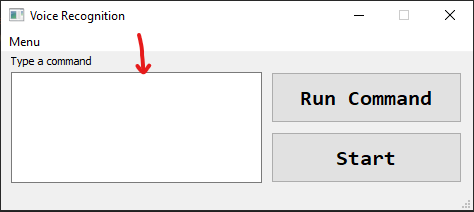
1. Sub-sentence b is empty; thus, no keyword is detected in it.
2. Sub-sentence c gets scanned by parse\_text() and the word “play” is found. Since the word “play” has the same priority as “search,” additional processes are conducted in order for the program to execute the intended command. parse\_text() looks whether “play” or “search” comes first, and since the latter comes first, it checks the next keyword which in this case is “spotify.”
3. play\_spotify() is invoked, and it intelligently finds the song search query and asks for the URI of the song through spotify API. It then sends a signal to play the song on a “computer” type device through the API.

# Program Manual

* Press the Start button to enable the hotkey which enables speech recognition to record your voice (shown in the next picture):



* Press the keys “k” + “l” to enable the voice command in the UI or type your command in the empty text box to do some actions with the program
* You can see the list of keywords and commands that can be used in our program in the help section in the menu bar.
* If you want multiple commands for the program (e.g. Open spotify, then play music then open microsoft word), you can speak or type “and”, “then”, or “and then” to enable multiple commands.
* You can also type the command as you would speak it in the “Type a command” box shown in the following picture:



# Discussion

## **6.1. Conclusion**

From what we have implemented, we feel that we’ve achieved our goals. While we were satisfied with what we have come up with, we could still improve on some aspect of the program; for instance, we designed the program with modularity in mind but its still not completely modular.

**6.2. Recommendation**

### **6.2.1. Offline Capabilities**

One of the bottlenecks that makes the user wait in our program is google’s speech recognition API. Theoretically, we can train a model of speech recognition and use another library in python to let us do the speech recognition offline.

### **6.2.2. Machine learning**

With machine learning, the process of parsing the commands would be completely different. We will no longer be simply using a bottom-up model which means that the program would be able to understand a wider set of sentence structures and correct misspoken words without even adding them to the list of possible misspoken words

# References

* <https://pypi.org/project/SpeechRecognition/>
* <https://youtu.be/RVMqB4W_EH4>
* <https://docs.python.org/3/library/index.html>
* <https://www.youtube.com/watch?v=eYJTcLBQKug>
* <https://developer.spotify.com/documentation/>
* <https://developers.google.com/youtube/v3/docs>

**Github Link :** [**https://github.com/alibanana/COMP6340-Final-Project-SR**](https://github.com/alibanana/COMP6340-Final-Project-SR)

**Link Video :** [**https://youtu.be/8N6OrvrYttY**](https://youtu.be/8N6OrvrYttY)