

목차

- 1. 프로젝트 설명**
 - a. 프로젝트 사용기술
 - b. 프로젝트 설명 및 개발 계기
 - c. 프로젝트 개발 과정
- 2. 프로그램 디자인 및 기능 설명**
 - a. 프로그램 디자인 설계 과정
 - b. 프로그램 기능 설명 및 설계과정
- 3. 버그발생 및 교체**
- 4. 프로젝트 느낀점**

1-a. 프로젝트 사용 기술

- 언어 : Python
- 프레임워크 : Pyside6
- 개발툴 : pycharm
- 프로젝트 관리 : git

1-b. 프로젝트 설명 및 개발 계기

설명 및 개발 계기 : 기존의 키보드 매크로 프로그램들은 사용법이 복잡하여 보다 간단한 프로그램을 만들고자 했습니다. 그리고 **Python**이 나름 촉망받는 언어이기도 하고 **Python**에 대해 기본적인 문법(**if**문, **for**문, **while**문등)들만 알고 있는 상태이기도 해서 연습도 좀 할겸 프로젝트를 하기로 마음먹었습니다.

1-c. 프로젝트 개발 과정

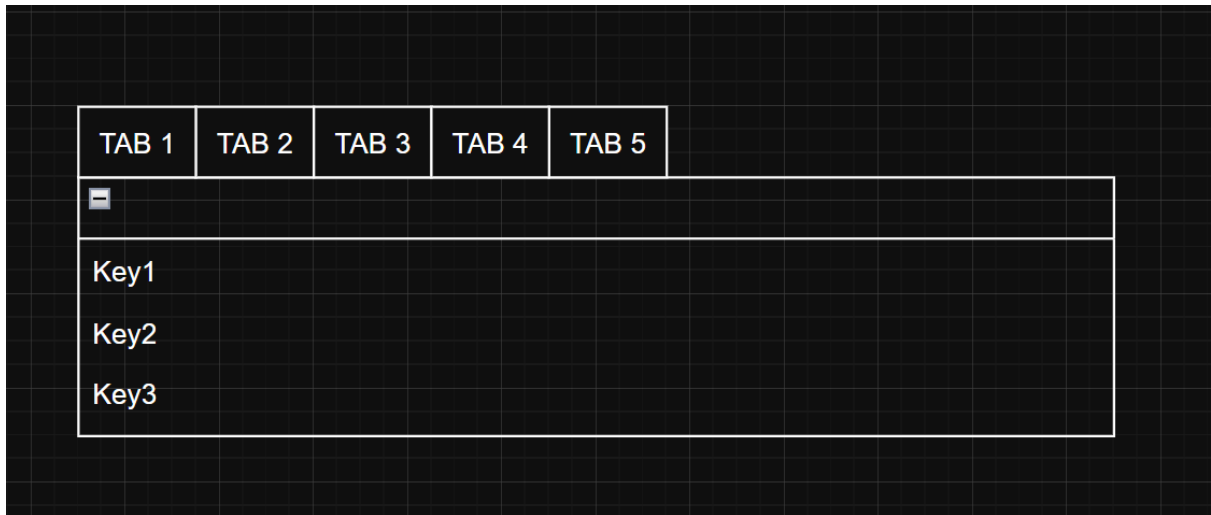
툴 및 프레임워크 선택 : 제가 매크로 프로그램을 만들려고 하는데 매크로 프로그램을 만든다는게 처음에는 어떤건지 잘 모르다가 구글링을 통하여 이러한 프로그램을 만드는게 **GUI프로그래밍**이라는 것을 알게 되었습니다. 그리고 **Python GUI** 프레임워크들 중 **Pyside**가 좀 더 세련되고 해당 **API**가 정리도 잘 되어 있어서 선택을 하게되었습니다.

프레임워크 API 공부 및 테스트 : 해당 프레임워크 사이트에 들어가서 튜토리얼을 직접 코딩해보면서 해당 프레임워크에 대한 감을 배우려고 노력하였습니다. 여러가지 기능을 테스트도 하고 모르는 것이 있으면 구글링이나 유튜브등을 보면서 공부 하였습니다.

프로그램 디자인 및 기능 설계 : 이제 프레임워크에 대한 기본적인 지식을 가지고 어떻게 프로그램을 디자인을 할지 해당 디자인에 어떤 기능을 어떻게 추가할 지에 대해 생각을 하였고 개발도중에 다시 새로운 기능들을 배워야 할 필요가 있는경우 해당 기능을 다시 공부를 하여 프로그램 기능을 구현하였습니다.

2-a. 프로젝트 디자인 설계 과정

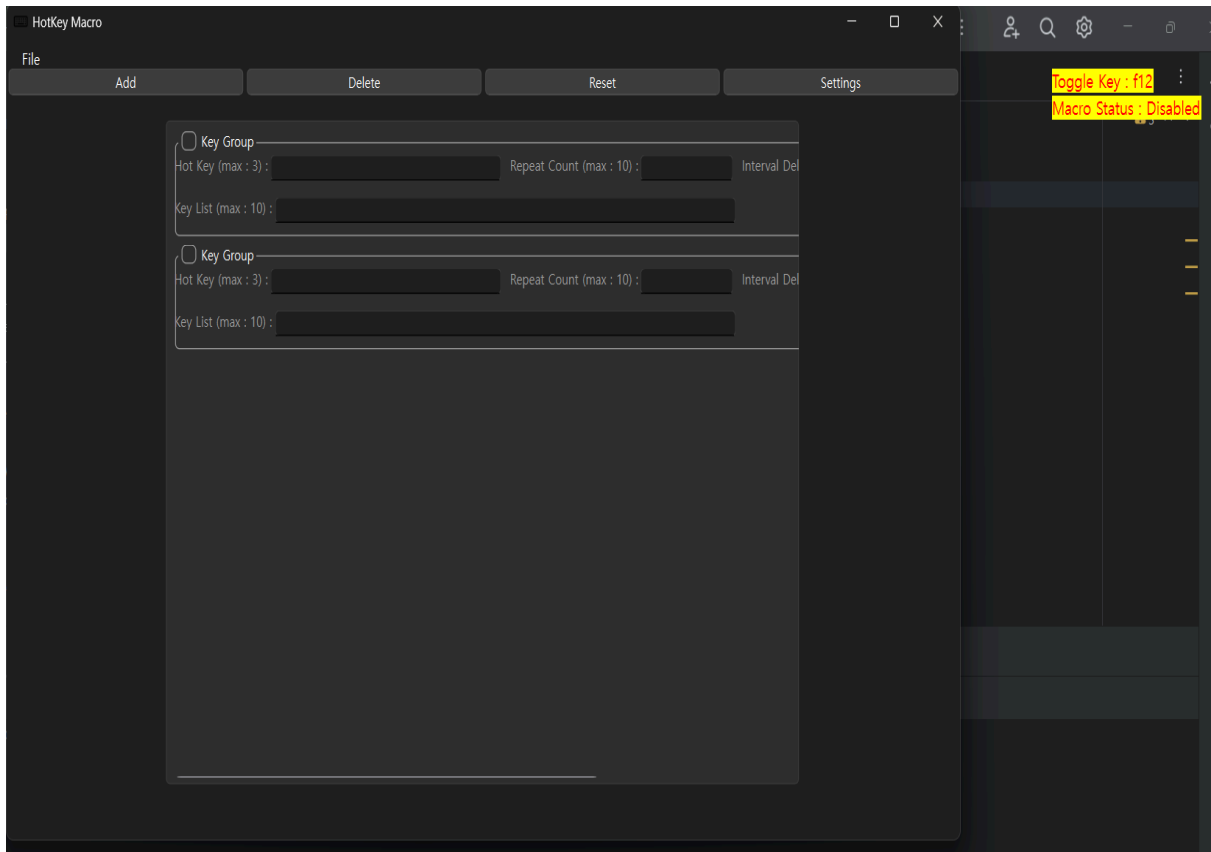
초기 디자인 : 초기 디자인은 약간 무식하게 설계를 하였습니다. 우선 다음과 같이 생각을 하였습니다.



설명 : 각 탭에 키를 저장하여 키보드 리스너가 전체 탭을 확인하면서 해당 키가 눌렸는지 안눌렸는지에 대해 체크하는 디자인을 설계하였습니다.

위의 디자인 문제점 : 위의 디자인에서 문제점은 특정 탭에 키를 부분적으로 저장을 하면 불필요한 다른 저장되지 않은 탭또한 검색을 해야하며 탭을 일일이 수동적으로 추가해야 한다는 불편함이 있습니다.

디자인 문제점 해결방법 : 위와 같은 문제를 해결하기 위해서 키를 저장하는 어떤 그룹을 동적으로 필요할때마다 생성해나가야 한다고 생각했습니다. 그 결과 디자인은 다음과 같습니다.

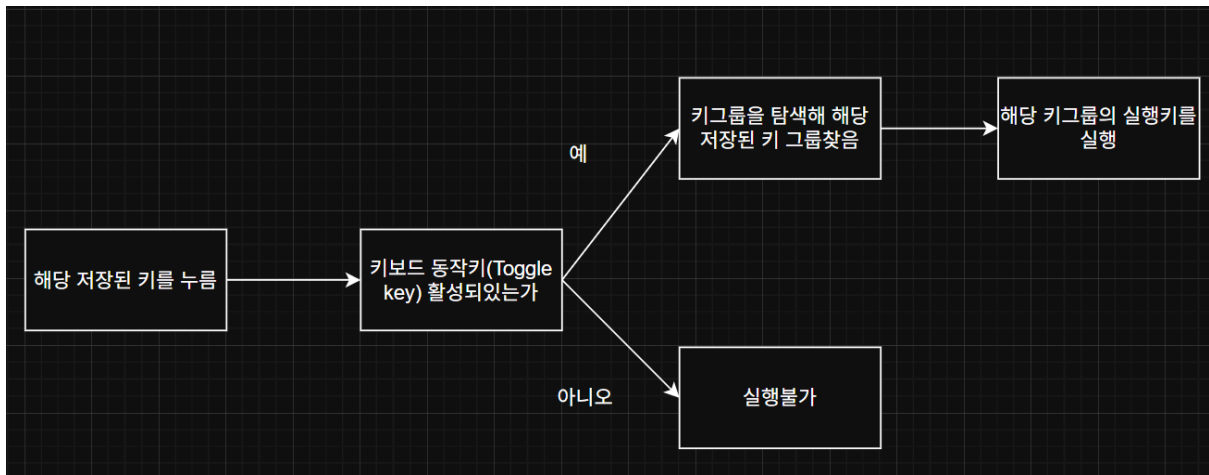


따라서 다음과 같은 디자인을 기본으로 구성하였습니다.

2-b. 프로젝트 기능 설명 및 설계과정

키 입력 과정 설명 : 키 입력은 해당 키입력을 누르면 스레드가 실행되면서 키 입력을 감지하게 됩니다. 키 입력의 포커싱 이벤트가 끝나게 되면 해당 스레드는 종료가 됩니다.

키보드 리스너 동작 과정 :



각 버튼 설명 : 각 버튼에 대한 설명입니다.

- **Add 버튼** : 키 그룹을 추가 합니다.

```

class AddButton(QPushButton): 1 usage
    def __init__(self,title,parent):
        super().__init__(title,parent)
        self.setFocusPolicy(Qt.NoFocus)
        self.ClickTrigger()
    def ClickTrigger(self): 1 usage
        self.clicked.connect(self.ClickAction)
    def ClickAction(self):| 1 usage
        # print("add button clicked")
        from ___MainScrollArea___ import main_scroll_frame,v_box_layout
        from ___GroupBox___ import GroupBox

        group_box = GroupBox()
        GroupBox.num_of_group += 1
        GroupBox.group_list.append(group_box)
        main_scroll_frame.resize(GroupBox.width,GroupBox.height * GroupBox.num_of_group)

        group_box.resize(group_box.width,group_box.height)
        group_box.setTitle("Key Group")
        v_box_layout.addWidget(group_box)

        # print("cur group num : ",len(GroupBox.group_list),GroupBox.num_of_group)
  
```

위와 같이 생성된 키그룹을 배열에 저장합니다.

- **Delete 버튼** : 전체 키그룹 배열에서 해당 키그룹을 삭제합니다.

```
class DeleteButton(QPushButton): 1 usage
    def __init__(self, title, parent):
        super().__init__(title, parent)
        self.setFocusPolicy(Qt.NoFocus)
        self.ClickTrigger()
    def ClickTrigger(self): 1 usage
        self.clicked.connect(self.ClickAction)
    def ClickAction(self): 1 usage
        # print("delete button clicked")
        from __MainScrollArea__ import main_scroll_frame, v_box_layout
        from __GroupBox__ import GroupBox

        total_checked = 0
        for_delete_groups = []
        for group in GroupBox.group_list:
            if group.isChecked() == True:
                total_checked += 1
                for_delete_groups.append(group)

                group.hot_key_line.hot_key_list.clear()

        for group in for_delete_groups:
            GroupBox.group_list.remove(group)
            group.deleteLater()

        GroupBox.num_of_group -= total_checked
        main_scroll_frame.resize(GroupBox.width, GroupBox.height * GroupBox.num_of_group)
```

해당 체크된 키그룹이 있는지 배열을 통해 탐색한 뒤에 삭제를 합니다. 그런다음 **deletelater**함수를 사용하여 **메모리 누수를 방지**해줍니다.

- **Reset 버튼** : 모든 키 그룹을 지웁니다. 전체 키그룹 배열이 초기화 됩니다.

```

class ResetButton(QPushButton): 1 usage
    def __init__(self, title, parent):
        super().__init__(title, parent)
        self.setFocusPolicy(Qt.NoFocus)
        self.ClickTriiger()
    def ClickTriiger(self): 1 usage
        self.clicked.connect(self.ClickAction)
    def ClickAction(self): 1 usage
        self.warning_message = QMessageBox()

        self.warning_message.setWindowTitle("Warning")
        self.warning_message.setWindowIcon(QIcon("./Images/Warning.png"))
        self.warning_message.setText("Do you really want to reset?")
        self.warning_message.setIcon(self.warning_message.Icon.Warning)

        self.warning_message.addButton(self.warning_message.StandardButton.Yes)
        self.warning_message.addButton(self.warning_message.StandardButton.No)

        self.result = self.warning_message.exec()
        if self.result == self.warning_message.StandardButton.Yes:
            self.YesButtonAction()
        elif self.result == self.warning_message.StandardButton.No:
            self.NoButtonAction()

def YesButtonAction(self): 1 usage
    # print("yes button clicked")

    from __MainScrollArea__ import main_scroll_frame, v_box_layout
    from __GroupBox__ import GroupBox

    GroupBox.num_of_group = 0
    main_scroll_frame.resize(0,0)

    for i in GroupBox.group_list:
        i.hot_key_line.hot_key_list.clear()
        if hasattr(i.hot_key_line, "hot_key_input_thread") == True:
            i.hot_key_line.hot_key_input_thread.terminate()
            i.hot_key_line.hot_key_input_thread.wait(1)
        i.repeat_count_line.repeat_count = ""
        if hasattr(i.repeat_count_line, "repeat_count_input_thread") == True:
            i.repeat_count_line.repeat_count_input_thread.terminate()
            i.repeat_count_line.repeat_count_input_thread.wait(1)

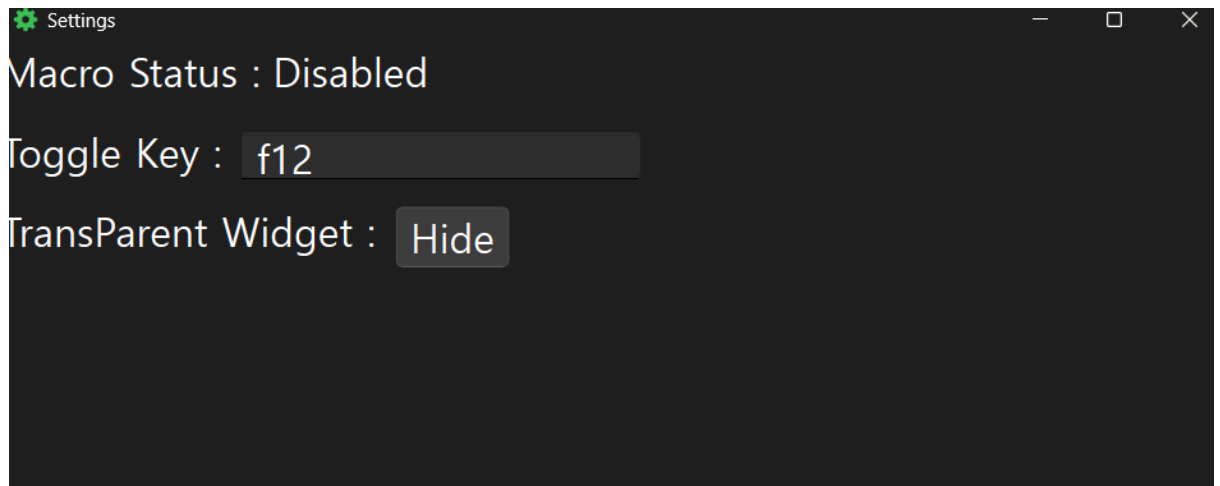
    for i in GroupBox.group_list:
        i.deleteLater()

    GroupBox.group_list.clear()

```

위와 같이 초기화 버튼을 누르면 ‘정말로 누르시겠습니까’라는 문구가 뜨고 ‘예’ 버튼을 누르면 위와 같이 모든 키그룹을 삭제합니다.

- **Setting 버튼** : 프로그램의 설정 버튼입니다. 다음과 같이 매크로를 실행할 수 있는지에 대한 상태, 토글 키, 투명위젯(노란색 화면)을 숨기거나 보이게 할 수 있습니다.



```
class ToggleKeyLineThread(threading.Thread): 2 usages
    def __init__(self):
        super().__init__(daemon=True)
        self.state = True
        self.delay = 0.01
        self.key_hook = None
    def run(self):
        self.key_hook = keyboard.on_press(callback=self.OnPress)

        while self.state:
            time.sleep(self.delay)
    def OnPress(self, event): 1 usage
        setting_widget.toggle_key_input_line.setReadOnly(True)

        if event.name in allowed_key_list:
            setting_widget.toggle_key_input_line.setText(event.name)
            setting_widget.toggle_key_input_line.key = event.name
    def stop(self): 5 usages (4 dynamic)
        self.state = False
```

위의 코드는 **Toggle key**를 변경하고자 누르면 해당 쓰레드가 실행되면서 키 입력을 감지하고 키 입력 포커싱아웃이 되면 쓰레드가 종료됩니다.

- **파일 저장** : 아래의 키그룹의 설정을 파일로 저장할 수 있습니다.

Key Group ☒

Hot Key (max : 3) : a + s Repeat Count (max : 10) : 1 Interval Del

Key List (max : 10) : 1 + 2 + 3

Key Group ☒

Hot Key (max : 3) : q + w Repeat Count (max : 10) : 1 Interval Del

Key List (max : 10) : 4 + 5 + 6

이를 구현한 코드는 다음과 같습니다.

```

def SaveFile(self,result): 1 usage
    if result == 1:
        print("save activated")
        file_info = QFileInfo(str(self.file_dialog.selectedFiles()[-1]))
        file_name = file_info.fileName()

        is_exist = self.CheckFileExist(file_name,file_info)
        if is_exist:
            message = QMessageBox()
            message.setWindowTitle("Error")
            message.setWindowIcon(QIcon("Error.png"))
            message.setText("file already exists")
            message.exec()
            return

        if ".mco" not in file_name:
            file = QFile("{}.{ {}".format(*args, file_name,"mco"))
            is_openable = False
            is_openable = file.open(QIODeviceBase.ReadWrite)

            if is_openable == False:
                print("file open error")
                read_err_msg = QMessageBox()
                read_err_msg.setWindowTitle("Error")
                read_err_msg.setWindowIcon(QIcon("Error.png"))
                read_err_msg.setText("File open error")
                read_err_msg.exec()
                file.close()
            return

```

```

else:
    file = QFile(file_name)
    is_openable = False
    is_openable = file.open(QIODeviceBase.ReadWrite)

    if is_openable == False:
        print("file open error")
        read_err_msg = QMessageBox()
        read_err_msg.setWindowTitle("Error")
        read_err_msg.setWindowIcon(QIcon("Error.png"))
        read_err_msg.setText("File open error")
        read_err_msg.exec()
        file.close()
        return

is_readable = self.ReadGroupCount(GroupBox.group_list, GroupBox.num_of_group)
if is_readable == True:
    pass
else:
    read_err_msg = QMessageBox()
    read_err_msg.setWindowTitle("Error")
    read_err_msg.setWindowIcon(QIcon("Error.png"))
    read_err_msg.setText("Read Group Error\n"
                        "please input data or check data")
    read_err_msg.exec()
    file.close()
    if ".mco" not in file_name:
        file.remove("{}.{ {}".format(*args, file_name, "mco"))
    else:
        file.remove(file_name)

```

위와 같이 각 키그룹에 대한 정보들을 불러와 저장하게 됩니다.
그런 다음, 다음과 같이 파일 형식에 맞게 데이터를 쓰게 됩니다.

```

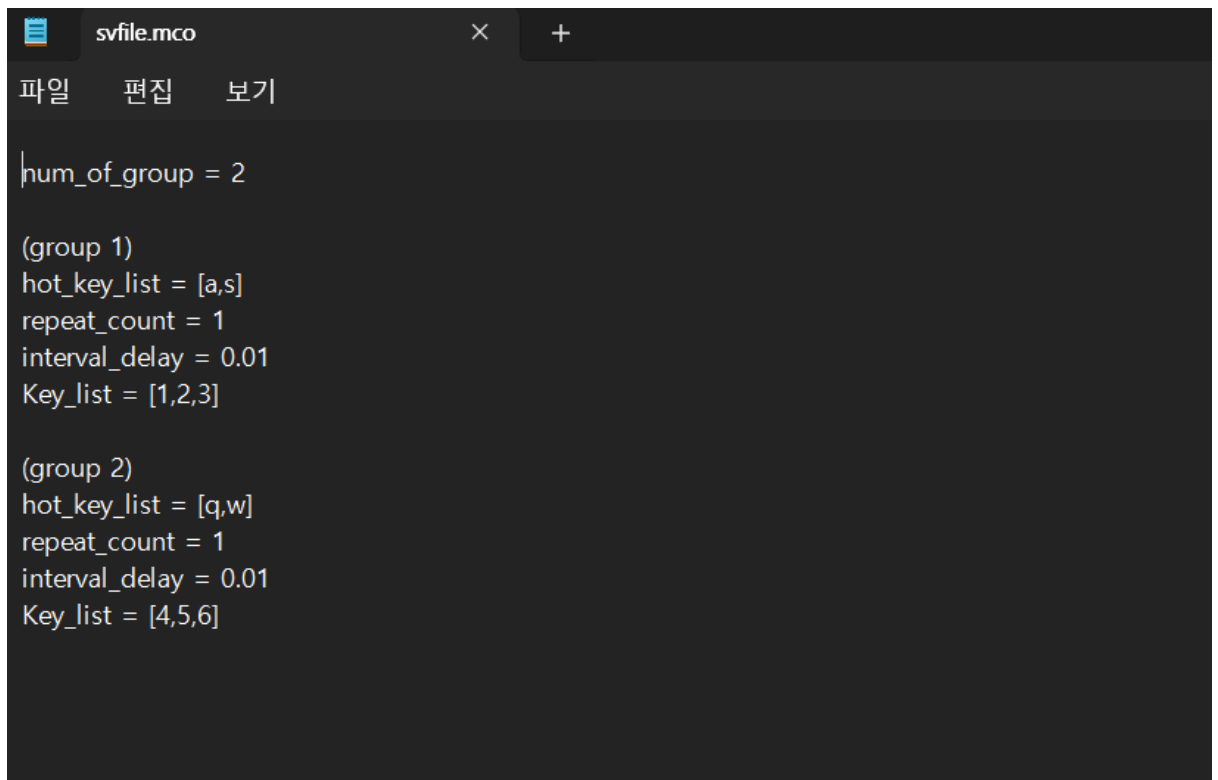
def WriteData(self, file, group_list): 1 usage
    file.write(QByteArray("num_of_group = {}\r\n".format(len(group_list))))
    file.write(QByteArray("\r\n"))

    for group_index, group in enumerate(group_list, start=1):
        file.write(QByteArray("(group {})\r\n".format(group_index)))

        file.write(QByteArray("hot_key_list = ["))
        hot_key_list = group.hot_key_line.hot_key_list
        for hot_key_index, hot_key in enumerate(hot_key_list, start=1):
            if len(hot_key_list) == hot_key_index:
                if hot_key == ".":
                    file.write(QByteArray("full stop"))
                    file.write(QByteArray("]\r\n"))
                elif hot_key == ",":
                    file.write(QByteArray("comma"))
                    file.write(QByteArray("]\r\n"))
                else:
                    file.write(QByteArray(str(hot_key)))
                    file.write(QByteArray("]\r\n"))
            else:
                if hot_key == ".":
                    file.write(QByteArray("full stop"))
                    file.write(QByteArray(" "))
                elif hot_key == ",":
                    file.write(QByteArray("comma"))
                    file.write(QByteArray(" "))
                else:
                    file.write(QByteArray(str(hot_key)))
                    file.write(QByteArray(" "))

```

그리고 다음과 같은 형태의 파일로 저장이 됩니다.



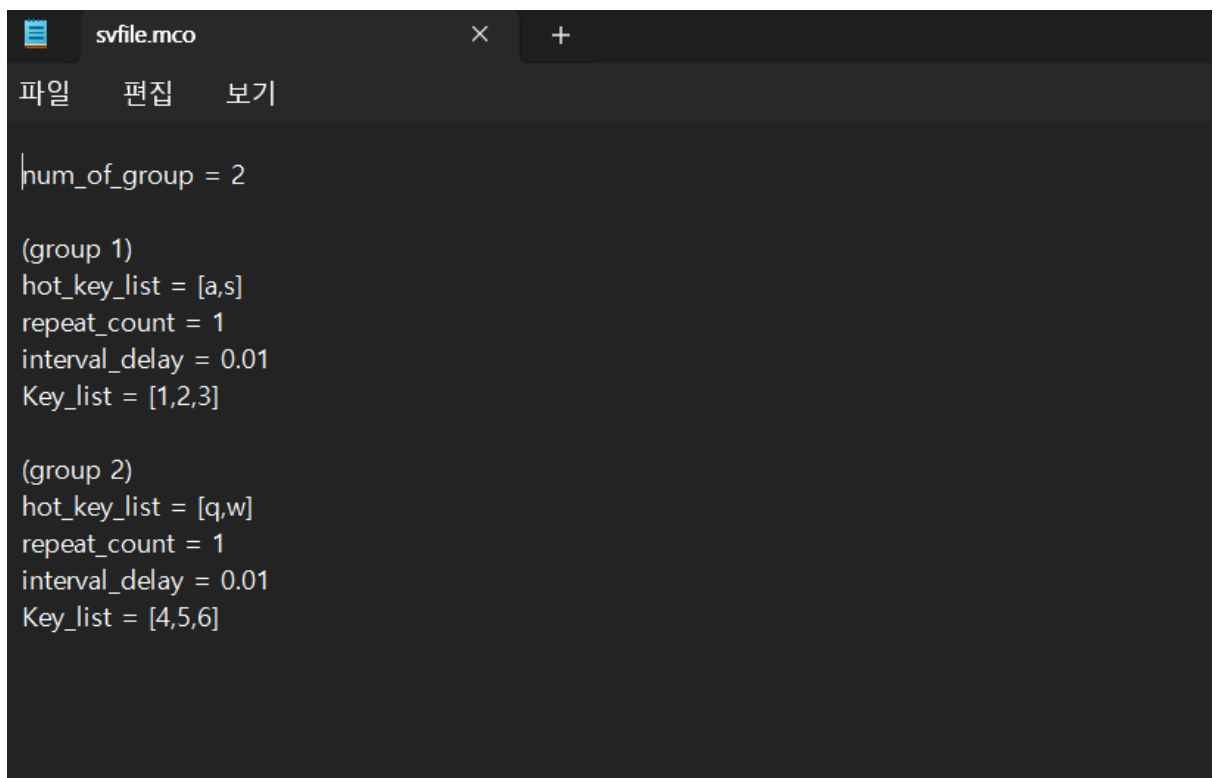
```
svfile.mco
파일  편집  보기

num_of_group = 2

(group 1)
hot_key_list = [a,s]
repeat_count = 1
interval_delay = 0.01
Key_list = [1,2,3]

(group 2)
hot_key_list = [q,w]
repeat_count = 1
interval_delay = 0.01
Key_list = [4,5,6]
```

- **파일 로드** : 위와 같이 저장된 파일을 불러오는 역할을 합니다.



```
svfile.mco
파일  편집  보기

num_of_group = 2

(group 1)
hot_key_list = [a,s]
repeat_count = 1
interval_delay = 0.01
Key_list = [1,2,3]

(group 2)
hot_key_list = [q,w]
repeat_count = 1
interval_delay = 0.01
Key_list = [4,5,6]
```

위의 저장된 파일은 다음과 같이

num_of_group(띄어쓰기)(group1)(띄어쓰기)(group2) ...로
구성이 되고 띄어쓰기 까지 문자열을 구분해서 읽으면서

해당파일에 설정된 상태를 읽어와 키그룹에 추가합니다. 다음은 해당 기능을 구현한 코드입니다.

```
def LoadFile(self,result):1 usage
    readed_data = ""
    if result == 1:
        print("Load activated")
        file_info = QFileInfo(str(self.file_dialog.selectedFiles()[-1]))
        file_name = file_info.fileName()

        file = QFile(file_name)
        is_openable = file.open(QIODeviceBase.ReadOnly)
        if is_openable == False:
            open_err_msg = QMessageBox()
            open_err_msg.setWindowTitle("Error")
            open_err_msg.setText("File open error")
            open_err_msg.setWindowIcon(QIcon("Error.png"))
            open_err_msg.exec()
            file.close()
            return

        try:
            while file.atEnd() == False:
                readed_data += file.read(1)
        except:
            open_err_msg = QMessageBox()
            open_err_msg.setWindowTitle("Error")
            open_err_msg.setText("File read error")
            open_err_msg.setWindowIcon(QIcon("Error.png"))
            open_err_msg.exec()
```

```

each_data = ""
global num_of_group
num_of_group = 0
global hot_key_lists
hot_key_lists = []
global repeat_counts
repeat_counts = []
global interval_delays
interval_delays = []
global key_lists
key_lists = []

for char in readed_data:
    each_data += char

    if "\r\n" in each_data:
        is_readable = self.ReadFileNumOfGroup(each_data)
        if is_readable == True:
            pass
        elif is_readable == False:
            pass
        elif is_readable == None:
            err_msg = QMessageBox()
            err_msg.setWindowTitle("Error")
            err_msg.setText("num_of_group read error")
            err_msg.setWindowIcon(QIcon("Error.png"))
            err_msg.exec()
            file.close()

```

위의 코드와 같이 파일을 읽어들인뒤, 해당 데이터들을 저장합니다.

```

each_data = ""
global num_of_group
num_of_group = 0
global hot_key_lists
hot_key_lists = []
global repeat_counts
repeat_counts = []
global interval_delays
interval_delays = []
global key_lists
key_lists = []

for char in readed_data:
    each_data += char

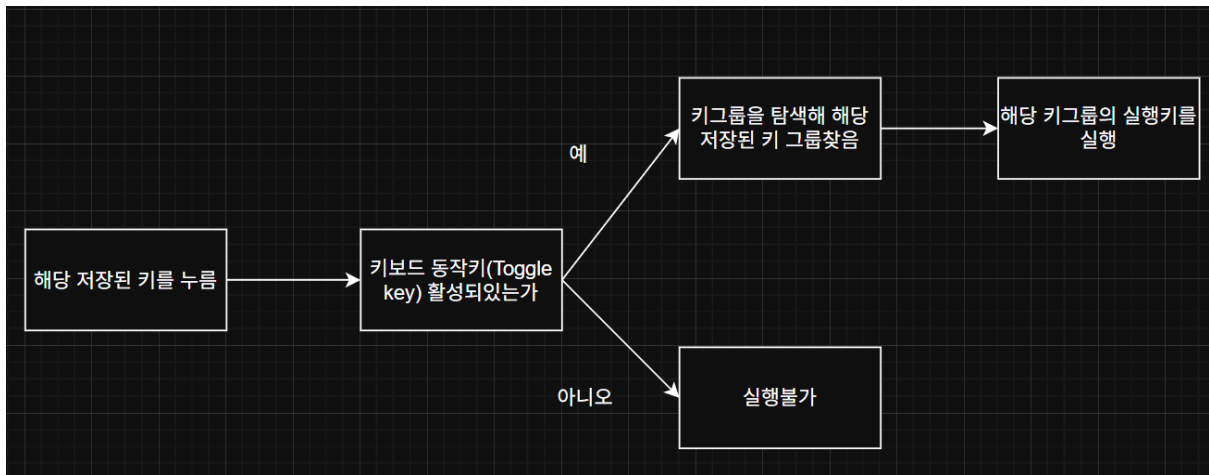
    if "\r\n" in each_data:
        is_readable = self.ReadFileNumOfGroup(each_data)
        if is_readable == True:
            pass
        elif is_readable == False:
            pass
        elif is_readable == None:
            err_msg = QMessageBox()
            err_msg.setWindowTitle("Error")
            err_msg.setText("num_of_group read error")
            err_msg.setWindowIcon(QIcon("Error.png"))
            err_msg.exec()
            file.close()

```

저장된 데이터들은 위의 코드를 통해 키그룹을 추가합니다.

키 입력 과정 설명 : 키 입력은 해당 키입력을 누르면 스레드가 실행되면서 키 입력을 감지하게 됩니다. 키 입력의 포커싱 이벤트가 끝나게 되면 해당 스레드는 종료가 됩니다.

키보드 리스너 동작 과정 :



3. 버그 발생 및 교체

설명 : 이렇게 만든 뒤에 테스트를 진행해보니 키보드 연타시에 키가 계속 눌러지는 버그가 발생하였습니다. 코드가 잘못되어 있는지 확인해보았지만 별다른 이상이 없는것 같았습니다.

그래서 한번 다른 키보드 모듈을 불러와 테스트를 해보았더니 별다른 이상이 없었고 본인이 사용한 키보드 모듈(**keyboard**)을 테스트 해보았더니 해당 **keyboard listener** 에 버그가 있단 걸 알았습니다.

keyboard 모듈 제작자가 해당 모듈 설명에서 더 이상 업데이트를 안한다고 하니 결국 키보드 모듈을 교체해야 하기로 결정하였습니다.

키보드 리스너는 **ctype**의 **win32** 키보드 리스너를 사용하기로 결정하였고,
기존의 키 시스템과의 호환성을 위해 매핑 코드를 삽입하였습니다.
아래의 코드는 해당 매핑을 구현한 코드입니다.

```

def ReadHotKeyList(self, group): 1 usage
    key_list = group.hot_key_line.hot_key_list
    correct_count = 0

    try:
        for key, value in mapped_numpad_vk_codes.items():
            if ctypes.windll.user32.GetAsyncKeyState(key) & 0x8000 and value in key_list:
                correct_count += 1

        for key in key_list:
            scList = keyboard.key_to_scan_codes(key)
            # print(f"key name : {key}          scan code list : {scList}")
            try:
                for scanCode in scList:
                    vkCode = ctypes.windll.user32.MapVirtualKeyA(scanCode, 1)
                    if ctypes.windll.user32.GetAsyncKeyState(vkCode) & 0x8000:
                        correct_count += 1
                        break
            except:
                return False
        print(correct_count)
        if correct_count == len(key_list) and correct_count != 0 and ___Keys___.toggle_key not in key_list:
            return True
        else:
            return False
    except:
        return False

```

위의 코드를 보시면 기존의 키 코드를 다시 키보드 스캔코드로 변환을 합니다. **scList = keyboard.key_to_scan_codes(key)**가 이 코드입니다.

이렇게 변환된 스캔코드를 **MapVirtualKeyA** 함수를 이용하여 **VK** 키코드로 변환을 합니다. **VK** 키코드로 변환을 하는 이유는 **win32** 키보드 리스너를 사용하기 위함입니다.

이렇게 해주면 **win32** 키보드 리스너가 해당 키를 인식을 하게되어 동작을 하게 됩니다.

즉, 기존 키코드->스캔코드->VK코드 이렇게 매핑함으로서 이러한 버그문제를 해결하였습니다.

4. 프로젝트 느낀점

설명 : 간단한 매크로 프로그램을 만드느거니 제작기간이 짧게 걸릴거라 생각했지만, 꽤 오래걸렸습니다. 개발에 대한 개념도 없고 파이썬 언어도 제대로 모르면서 맨땅에 헤딩하는 느낌으로다가 개발을 하였습니다. 개발 중간중간에 필요한 개념(**스레드, API** 등, 클래스, 상속 등)등을 공부하면서 만든거라 힘들었습니다. 포기할까도

생각했지만 포기하지 않고 수많은 노력끝에 만들게 되었습니다. 결국
만들게 되니 기분이 좋았습니다.