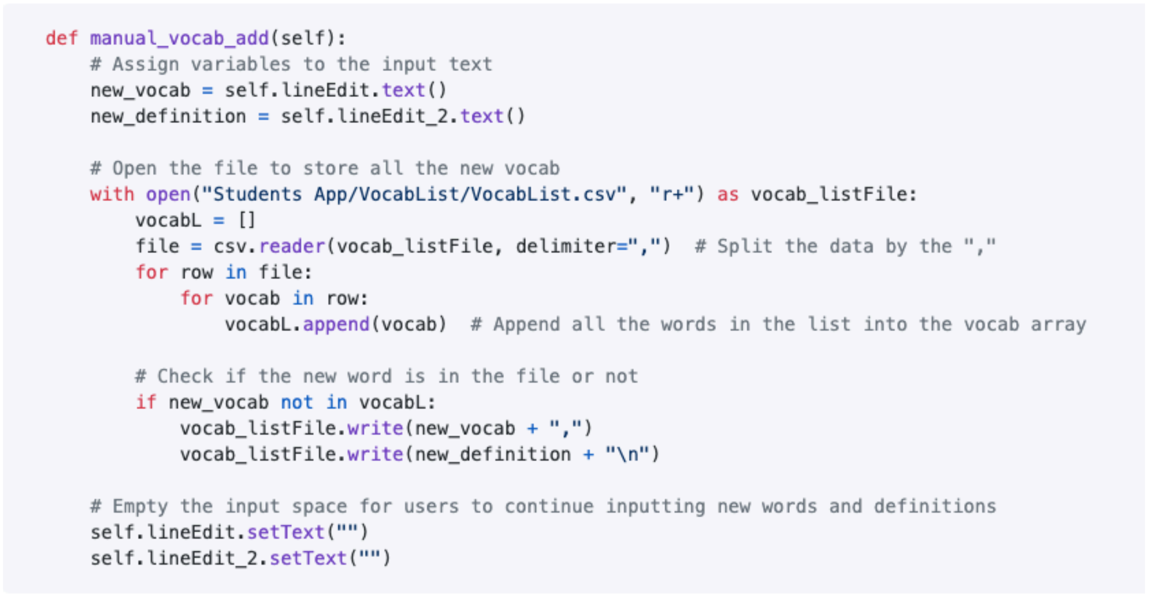
Criterion C: Development

List of Techniques

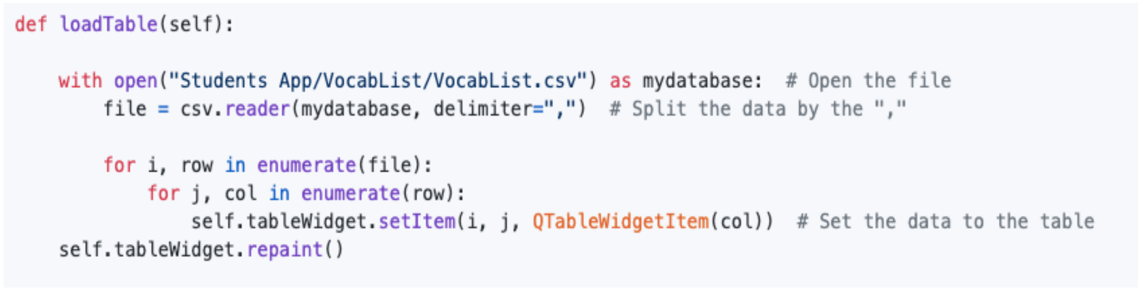
1. **Database**

In the Japanese Learning system, it will store the vocabulary, its definition, and quiz. The users can add new vocabulary manually by writing inputted vocabulary and its definition into line Edits, which then be stored in *csv* file, so that the program will load data from it. If the vocabulary already exists in the file so there is no need to add the word into the file.



```python
def manual_vocab_add(self):
    # Assign variables to the input text
    new_vocab = self.lineEdit.text()
    new_definition = self.lineEdit_2.text()

    # Open the file to store all the new vocab
    with open("Students App/VocabList/VocabList.csv", "r+") as vocab_listFile:
        vocabL = []
        file = csv.reader(vocab_listFile, delimiter=",")  # Split the data by the ","
        for row in file:
            for vocab in row:
                vocabL.append(vocab)  # Append all the words in the list into the vocab array

        # Check if the new word is in the file or not
        if new_vocab not in vocabL:
            vocab_listFile.write(new_vocab + ",")
            vocab_listFile.write(new_definition + "\n")

    # Empty the input space for users to continue inputting new words and definitions
    self.lineEdit.setText("")
    self.lineEdit_2.setText("")
```

The table inside the Vocab List window also need to represent the data in the *.csv* file to show the users. I will first read the data, and implement two loops, one for assigning the index row and the text of each item in the database file, one for taking out the item and the index col of each item in a row. Lastly, write data into cells of the table through "tableWidget.setItem(row, col, content)" function.



```python
def loadTable(self):

    with open("Students App/VocabList/VocabList.csv") as mydatabase:  # Open the file
        file = csv.reader(mydatabase, delimiter=",")  # Split the data by the ","

        for i, row in enumerate(file):
            for j, col in enumerate(row):
                self.tableWidget.setItem(i, j, QTableWidgetItem(col))  # Set the data to the table
    self.tableWidget.repaint()
```

The enumerate(file) function returns both the iterable, combined with the index of that value.

The user can manually delete the vocabulary. Firstly, it will get the row of selected cell in the table.

```python
global cell_row
selected_cell = self.tableWidget.selectedItems()

# Get the row of the selected cell in the QTable
for item in selected_cell:
    cell_row = item.row()
```

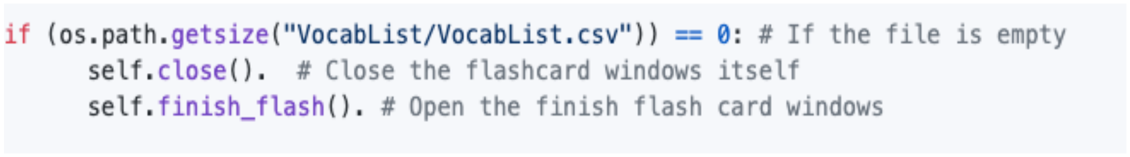
I will add all words and its definition into an array, then check if the line in the file starts with the deleted vocab, if it does not, append it into the *output array* so the deleted vocab will no longer exist in the *output* array. Finally, write the *output array* into the database file.

```python
with open("Students App/VocabList/VocabList.csv", "r") as out_file:
    file = csv.reader(out_file, delimiter=",")  # Split the data by the ","
    vocabArray = []
    output = []

    for word in file:
        vocabArray.append(word)  # Add all the words in the list into the array

with open("Students App/VocabList/VocabList.csv", "r") as out_file_1:
    for line in out_file_1:
        if not line.startswith(vocabArray[cell_row][0]):  # if the line not startwith the String Input
            # append it to the output list => The name deleted will no longer in the list
            output.append(line)
    out_file_1.close()

f = open("Students App/VocabList/VocabList.csv", "w")
f.writelines(output)  # Write the output list to the Database file
f.close()
```

2. Flashcard

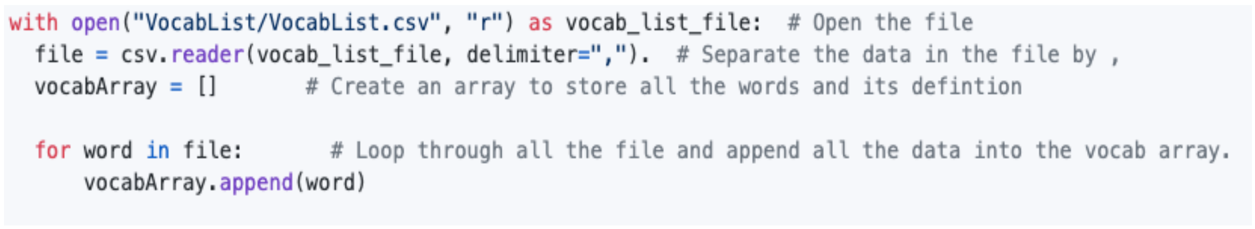
One of the priorities in the success criteria is implementing the flash card based on the vocabulary list they make. Thus, the database of the vocabulary list is used for the flash card.

Firstly, I will check if the vocab list is empty or not and open the vocabulary list file. If it is empty, there is no need to load the file and display the vocabulary and its definition there. Instead showing the finish flashcard window.



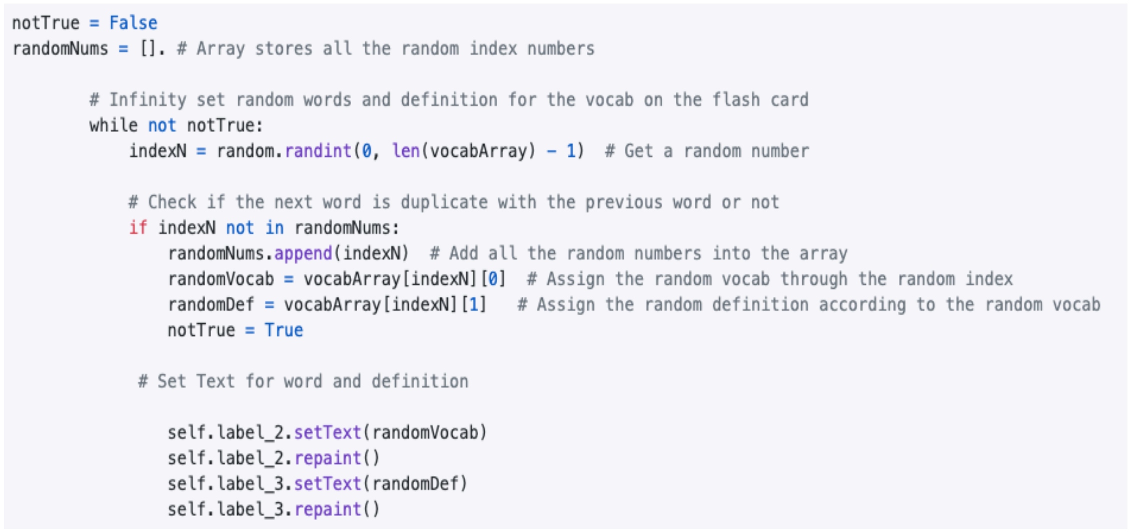
```
if (os.path.getsize("VocabList/VocabList.csv")) == 0: # If the file is empty
      self.close().  # Close the flashcard windows itself
      self.finish_flash(). # Open the finish flash card windows
```

Otherwise, open the vocabulary list file to load the data.



```
with open("VocabList/VocabList.csv", "r") as vocab_list_file:  # Open the file
  file = csv.reader(vocab_list_file, delimiter=",").  # Separate the data in the file by ,
  vocabArray = []       # Create an array to store all the words and its defintion

  for word in file:       # Loop through all the file and append all the data into the vocab array.
      vocabArray.append(word)
```

Generate a random number to take the index of random vocabulary inside the array. By generating list of random number within the range of vocabulary array length, the vocabulary can be taken randomly from the list as their indexes are random. Also, make sure that there are no duplicated random indexes appear. Lastly, display the text.

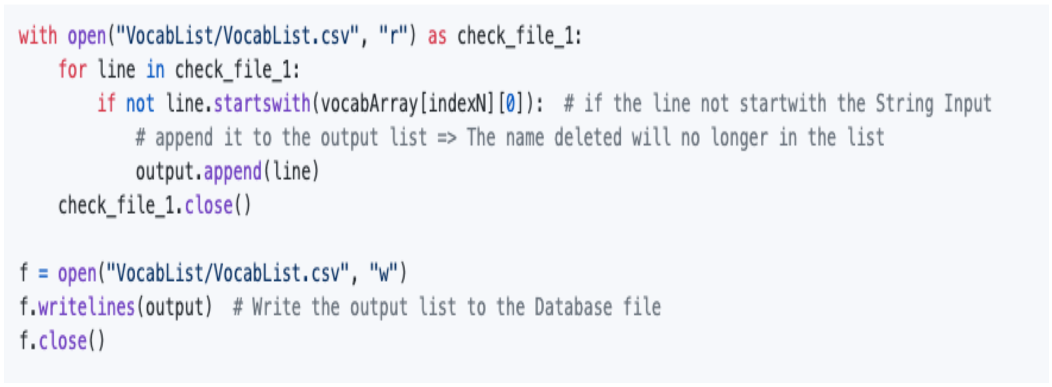


```
notTrue = False
randomNums = []. # Array stores all the random index numbers

        # Infinity set random words and definition for the vocab on the flash card
        while not notTrue:
            indexN = random.randint(0, len(vocabArray) - 1)  # Get a random number

            # Check if the next word is duplicate with the previous word or not
            if indexN not in randomNums:
                randomNums.append(indexN)  # Add all the random numbers into the array
                randomVocab = vocabArray[indexN][0]  # Assign the random vocab through the random index
                randomDef = vocabArray[indexN][1]   # Assign the random definition according to the random vocab
                notTrue = True

             # Set Text for word and definition

                self.label_2.setText(randomVocab)
                self.label_2.repaint()
                self.label_3.setText(randomDef)
                self.label_3.repaint()
```

Inside the flashcard window, there are two radio buttons:

- "I have known this word": The program will delete the word from the vocab list (as the users' requirement).

  Open the vocab file again and check if any lines inside the file do not start with the deleted vocab. If it does not, I will append to a new array. So, through this, the deleted vocab will not be in the array.

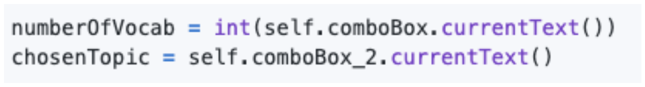


```
with open("VocabList/VocabList.csv", "r") as check_file_1:
    for line in check_file_1:
        if not line.startswith(vocabArray[indexN][0]):  # if the line not startwith the String Input
            # append it to the output list => The name deleted will no longer in the list
            output.append(line)
    check_file_1.close()

f = open("VocabList/VocabList.csv", "w")
f.writelines(output)  # Write the output list to the Database file
f.close()
```

- "I have not known this word": The program will just do the process of loading the flash card.

3. **Students App Quizz**

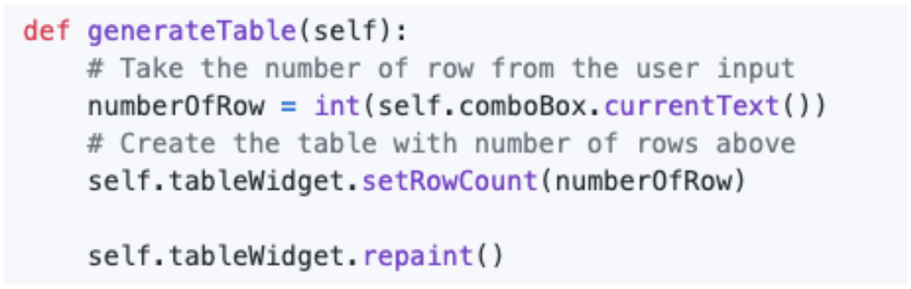
The number of vocab will determine how many quizzes to generate and the topic chosen will determine which data the program will take. It will be taken from the input that the user chose.



```
numberOfVocab = int(self.comboBox.currentText())
chosenTopic = self.comboBox_2.currentText()
```

***) Generate empty table with the inputted number of vocabs and rows**

As the user can choose the number of vocab in the quiz so that the program will generate table with the inputted number of vocab.



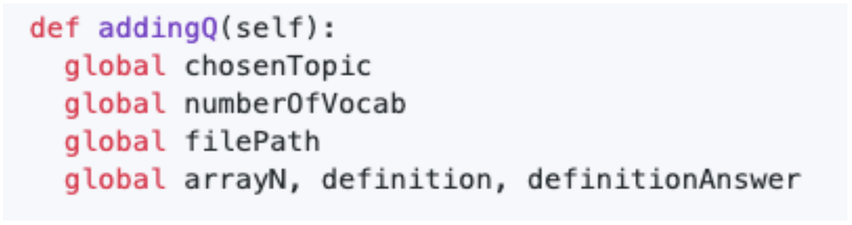
```
def generateTable(self):
    # Take the number of row from the user input
    numberOfRow = int(self.comboBox.currentText())
    # Create the table with number of rows above
    self.tableWidget.setRowCount(numberOfRow)

    self.tableWidget.repaint()
```

.tableWidget.setRowCount(numberOfRow) is the syntax for creating a table with the given number of *rows.*

***) Generate random questions**

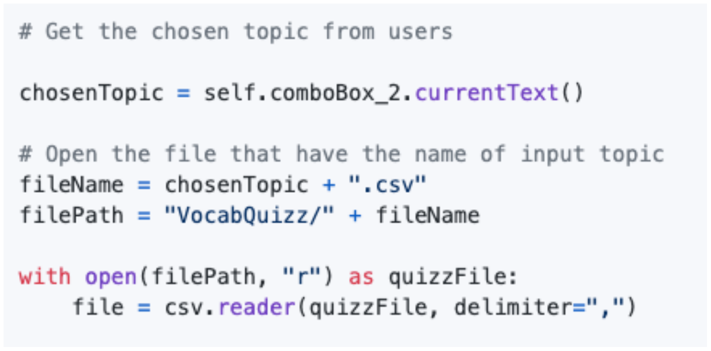
As I will use these variables for the other methods: *checkAnswers* and show the answers in the history windows not to duplicate

defining variables so that I will make the variables global.
The *global* keyword allows modify variables outside of the current scope.



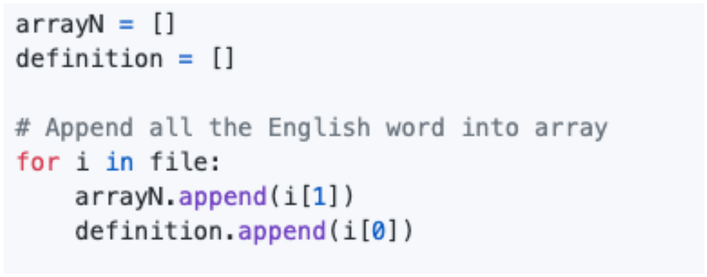
```
def addingQ(self):
  global chosenTopic
  global numberOfVocab
  global filePath
  global arrayN, definition, definitionAnswer
```

Get the topic chosen through the user input to generate vocabulary by topic. Then, concatenate the string with .csv to make the name of the file and assign to variable *fileName*. All the vocab quizz files are stored in the *VocabQuizz* folder so that
concatenating *VocabQuizz/* with the *fileName* will make up the Path to the file. By doing this, the program will get the data from the csv file that contain the vocabs sorted by topic.



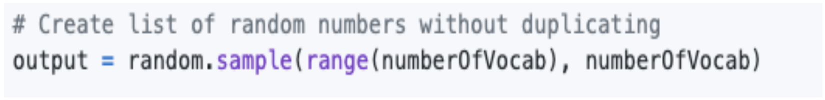
```
# Get the chosen topic from users

chosenTopic = self.comboBox_2.currentText()

# Open the file that have the name of input topic
fileName = chosenTopic + ".csv"
filePath = "VocabQuizz/" + fileName

with open(filePath, "r") as quizzFile:
    file = csv.reader(quizzFile, delimiter=",")
```

I will append all the English words inside the vocab file into an array to access them through indexes. Also, append the Japanese words in order to check the answers.



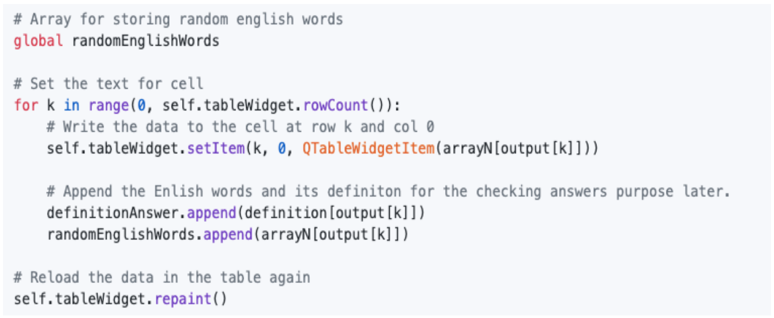
```python
arrayN = []
definition = []

# Append all the English word into array
for i in file:
    arrayN.append(i[1])
    definition.append(i[0])
```

As program randomly takes out the words from the database so that it is crucial to not duplicate the words taken out. I create an array that contains random numbers without duplicate. These numbers are the indexes of the words in the English and Japanese array created in the previous step. *random.sample(population, k)* generates a list of **unique** elements within the **range of population** and with the length of **k**.



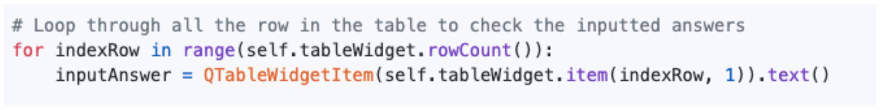
```python
# Create list of random numbers without duplicating
output = random.sample(range(numberOfVocab), numberOfVocab)
```

Next write data to the table with the random English words by using *tableWidget.setItem*(row,0, item) to write the item to the cell at row = row (in range of the number of vocab), col = 0. The reason is that the quiz will only display the English word for the students in the first column and the cells in the second column will be empty so that the users can input new words when taking the quiz.



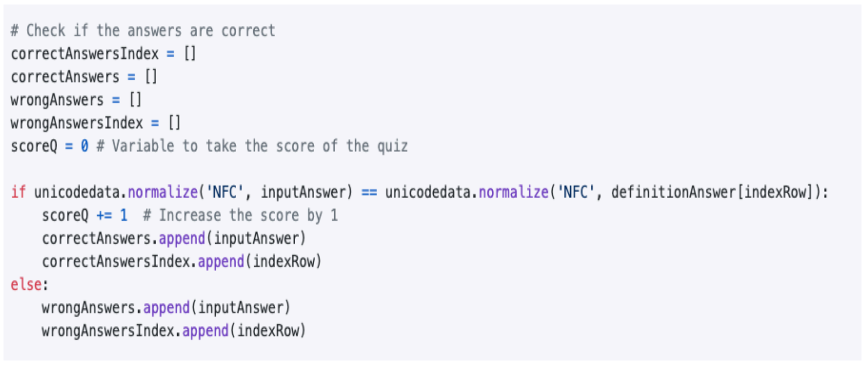
```python
# Array for storing random english words
global randomEnglishWords

# Set the text for cell
for k in range(0, self.tableWidget.rowCount()):
    # Write the data to the cell at row k and col 0
    self.tableWidget.setItem(k, 0, QTableWidgetItem(arrayN[output[k]]))

    # Append the Enlish words and its definiton for the checking answers purpose later.
    definitionAnswer.append(definition[output[k]])
    randomEnglishWords.append(arrayN[output[k]])

# Reload the data in the table again
self.tableWidget.repaint()
```

One success criteria of the program is that it will automatically check the answers and calculate the results for the students.
 I will loop through all the rows in the table, then use *QTableWidgetItem(self.tableWidget.item(row, col).text)* to take out the content of the element inside the cell at (row,col). As the inputted answers are all in the second column so that col = 1.



```
# Loop through all the row in the table to check the inputted answers
for indexRow in range(self.tableWidget.rowCount()):
    inputAnswer = QTableWidgetItem(self.tableWidget.item(indexRow, 1)).text()
```

As the inputted data is Japanese words that is non-ASCII text, so that Python cannot compare them. I used *unicodedata.normalize(form, unistr)* to return the normal form for the Unicode string. If the answer is correct, increase the *score* by 1, then append the correct Answers into *correctAnswers* array, else append all the wrong answer into the *wrongAnswers* list.



```
# Check if the answers are correct
correctAnswersIndex = []
correctAnswers = []
wrongAnswers = []
wrongAnswersIndex = []
scoreQ = 0 # Variable to take the score of the quiz

if unicodedata.normalize('NFC', inputAnswer) == unicodedata.normalize('NFC', definitionAnswer[indexRow]):
    scoreQ += 1  # Increase the score by 1
    correctAnswers.append(inputAnswer)
    correctAnswersIndex.append(indexRow)
else:
    wrongAnswers.append(inputAnswer)
    wrongAnswersIndex.append(indexRow)
```

Next, storing the students records into two files *fileHistoryNameCorrect*, which for correct answers and *fileHistoryNameWrong*, which for wrong answers for quiz. The finished quizz tables can take the data from these two files to display data into 2 tables inside the windows so that we need to store the correct and wrong answers. Also, by adding date taken the quiz will help teacher to keep track of it.

```python
def storeResult(self):

    # Take the date that the quiz is taken
    global today, d, fileHistoryNameWrong, fileHistoryPathWrong, fileHistoryNameCorrect, fileHistoryPathCorrect

    today = date.today()
    d = today.strftime("%d-%m-%Y")

    # Create the correct answers file with the name of current date
    fileHistoryNameCorrect = d + "-correctAns.csv"
    fileHistoryPathCorrect = "History File/" + fileHistoryNameCorrect

    # Create the wrong answers file with the name of current date
    fileHistoryNameWrong = d + "-wrongAns.csv"
    fileHistoryPathWrong = "History File/" + fileHistoryNameWrong

    # Write the correct answers to the file
    with open(fileHistoryPathCorrect, "w+") as historyFile:
        for i in range(len(correctAnswers)):
            historyFile.write(randomEnglishWords[correctAnswersIndex[i]] + ",")
            historyFile.write(correctAnswers[i] + "\n")

    # Write the wrong answers to the file
    with open(fileHistoryPathWrong, "w+") as historyFile_1:
        for m in range(0, len(wrongAnswers)):
            historyFile_1.write(randomEnglishWords[wrongAnswersIndex[m]] + ",")
            historyFile_1.write(wrongAnswers[m] + ",")
            historyFile_1.write(definitionAnswer[m + 1] + "\n")
```

The result window will display: The score of the quizz, Table of wrong answers, Table of correct answers. The *scoreQ* is the variable showing the result of the quiz that the user just finished. Also, as the user can choose the number of vocab to take so that showing the number of vocabs will help the teachers to keep track of students' performance. Furthermore, in the wrong answers, there will be 3 columns that are English words, inputted answers and the correct answers. From here the users can compare their answers with the correct answers.

```python
def loadResult(self):

  # Set the label for the result
  self.label_2.setText("Your Score is: " + str(scoreQ) + "/" + str(numberOfVocab))
  self.label_2.repaint()
  with open(fileHistoryPathWrong, "r+") as resultWrong:  # Open the file
      file = csv.reader(resultWrong, delimiter=",")  # Split the data by the ","

      numberOfRowWrong = len(wrongAnswers)
      # Create the table with number of rows above
      self.tableWidget.setRowCount(numberOfRowWrong)

      for i, row in enumerate(file):
          for j, col in enumerate(row):
              self.tableWidget.setItem(i, j, QTableWidgetItem(col))  # Set the data to the table

      # Reload the content of the table again
      self.tableWidget.repaint()
      # Erase the data inside the file

  with open(fileHistoryPathCorrect, "r+") as resultCorrect:  # Open the file
      file_1 = csv.reader(resultCorrect, delimiter=",")  # Split the data by the ","

      numberOfRowCorrect = len(correctAnswers)
      # Create the table with number of rows above
      self.tableWidget_2.setRowCount(numberOfRowCorrect)

      for i, row in enumerate(file_1):
          for j, col in enumerate(row):
              self.tableWidget_2.setItem(i, j, QTableWidgetItem(col))  # Set the data to the table

      # Erase the data inside the file
      # Reload the table
      self.tableWidget_2.repaint()
```

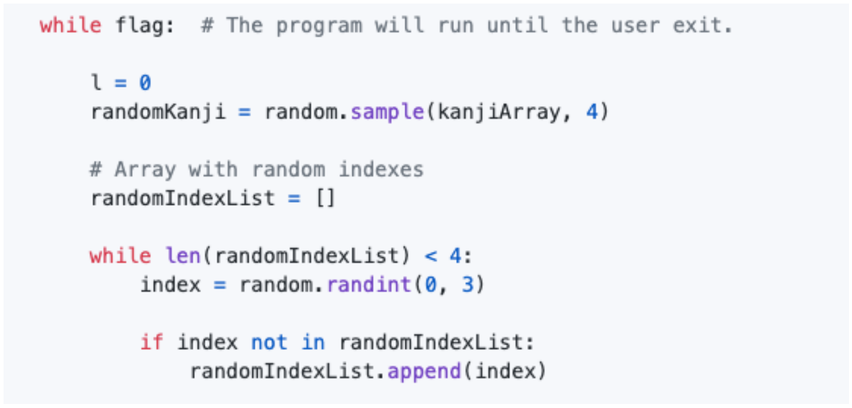
4. Kanji Learning Window

Kanji learning windows randomly generates kanji words from the database. In this part, as the client's requirement, there should be multiple-choice test without calculating the score. Also, it reduces the possibility that the users to just randomly pick up the correct answers.

First, generate 4 random kanji words from the kanji Array. The reason for 4 is that this is the multiple choices with 4 options.

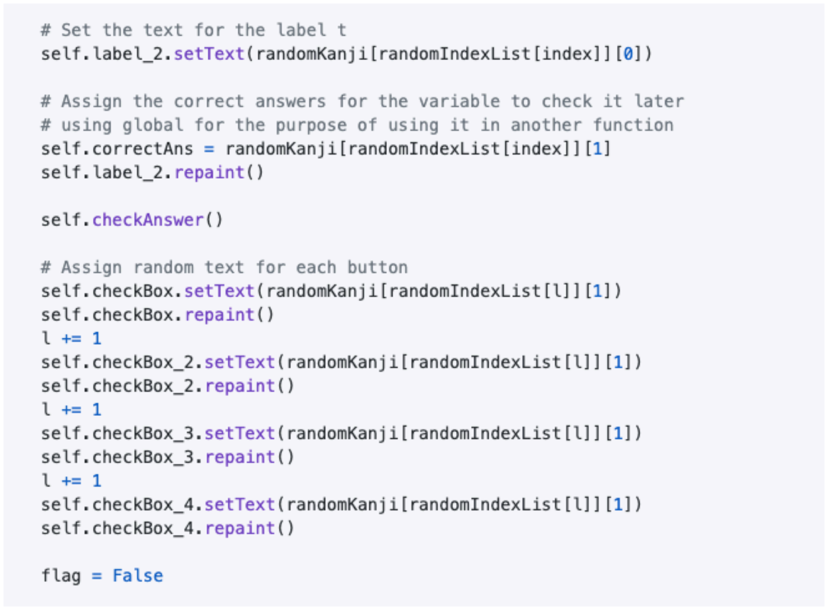
```python
randomKanji = random.sample(kanjiArray, 4)
```

The program will run every time the user presses the *Next* button and after generating the kanji, it will stop. I use while loop to implement the function as the number of times that the user presses the next button is unknown. Because the check boxes cannot be randomly taken, taking random text from the list and assign to the check boxes is a solution. The method is: generate 4 non-duplicated random numbers in range (0,3) as the length of the *kanjiArray* is 4 then append them all into array.



```python
while flag:  # The program will run until the user exit.

    l = 0
    randomKanji = random.sample(kanjiArray, 4)

    # Array with random indexes
    randomIndexList = []

    while len(randomIndexList) < 4:
        index = random.randint(0, 3)

        if index not in randomIndexList:
            randomIndexList.append(index)
```

Then simply set text for the label to display the kanji words and set text for each check box buttons to display the answers.



```python
# Set the text for the label t
self.label_2.setText(randomKanji[randomIndexList[index]][0])

# Assign the correct answers for the variable to check it later
# using global for the purpose of using it in another function
self.correctAns = randomKanji[randomIndexList[index]][1]
self.label_2.repaint()

self.checkAnswer()

# Assign random text for each button
self.checkBox.setText(randomKanji[randomIndexList[l]][1])
self.checkBox.repaint()
l += 1
self.checkBox_2.setText(randomKanji[randomIndexList[l]][1])
self.checkBox_2.repaint()
l += 1
self.checkBox_3.setText(randomKanji[randomIndexList[l]][1])
self.checkBox_3.repaint()
l += 1
self.checkBox_4.setText(randomKanji[randomIndexList[l]][1])
self.checkBox_4.repaint()

flag = False
```

Word count: 1327