```
########################
#Louis-Marie Noe NDOKI
#Jamie Emery
#Mahima Dalal
#Jay Davidge
#Joeph Maredith


########################

import tkinter as tk
import csv
import tkinter.messagebox as tm
import random as rdm
from tkinter import ttk
import tkinter.messagebox as tm
from tkinter import scrolledtext
import datetime
import tkinter
from tkinter import *


#//\\//\\//\\//\\/#
#QUESTIONIRE CLASS#
#//\\//\\//\\//\\/#

class Question:
    'Contains all information for a given question'

    def __init__(self, moduleCode, questionNumber, generatedQuestion= False):
        self.__questionNumber = questionNumber
        self.__generatedQuestion = generatedQuestion
        if not generatedQuestion:
            with open('test_'+ moduleCode +'.csv') as csvfile:
                rdr = csv.reader(csvfile, delimiter=',')
                for row in rdr:
                    if int(row[0]) == questionNumber:
                        self.__questionInformation = row[1]
                        self.__correctAnswer = row[2]
                        self.__incorrectAnswers = [row[3],row[4],row[5]]
        else:
            with open('testGen_'+ moduleCode +'.csv') as csvfile:
                rdr = csv.reader(csvfile, delimiter=',')
                for row in rdr:
                    if int(row[0]) == questionNumber:
                        self.__questionInformation = row[1]
                        self.__correctAnswer = row[2]
                        self.__incorrectAnswers = [row[3],row[4],row[5]]


    def __str__(self):
        a = 'Question Number: ' + str(self.__questionNumber) + ' | Question Information: ' + self.__questionInformation +  ' | Correct Answer: ' + self.__correctAnswer + ' | Incorrect Answers: ' + self.__incorrectAnswers[0] + ', '+ self.__incorrectAnswers[1] + ', '+ self.__incorrectAnswers[2]
        return aS

    def getGeneratedQuestion(self):
        return self.__generatedQuestion

    def getQuestionNumber(self):
        return self.__questionNumber

    def getQuestionInfo(self):
        return self.__questionInformation

    def getCorrectAnswer(self):
        return self.__correctAnswer

    def getIncorrectAnswers(self):
        return self.__incorrectAnswers

#//\\//\\\/#
#TEST CLASS#
#//\\//\\\/#
class Test:
    'Class that is used to run a test'

    def __init__(self, moduleCode, numberOfQuestions = 20):
        self.__moduleCode = moduleCode
        self.__currentQuestion = 1
        self.__currentMark = 0
        self.__questions = [Question(moduleCode, i) for i in range(1, numberOfQuestions + 1)]
        self.__selectedAnswers = []

    def getQuestionDetails(self, questionNumber = -1):
        if questionNumber == -1:
            return self.__questions[self.__currentQuestion - 1]
        else:
            return self.__questions[questionNumber - 1]

    def getNumberOfQuestions(self):
        return (len(self.__questions))

    def getCurrentQuestionNumber(self):
        return self.__currentQuestion

    def getSelectedAnswer(self, questionNumber):
        return self.__selectedAnswers[questionNumber - 1]

    def getCurrentMark(self):
        return self.__currentMark

    def incCurrentQuestion(self):
        self.__currentQuestion += 1

    def incCurrentMark(self):
        self.__currentMark += 1

    def addSelectedAnswer(self, answer):
        self.__selectedAnswers.append(str(answer))

    def generateQuestion(self):
```

```
111
112         return
113
114     def questionPersonalistaion(self):
115
116         return
117
118     def checkAnswer(self, providedAnswer, questionNumber = -1):
119         if questionNumber == -1:
120             question = self.__questions[self.__currentQuestion - 1]
121         else:
122             question = self.__questions[questionNumber - 1]
123
124         if str(providedAnswer) == str(question.getCorrectAnswer()):
125             return True
126         else:
127             return False
128
129     def saveMarks(self, fName, LName, filename='test_marks.csv'):
130
131         now = datetime.datetime.now()
132         nowDate = now.strftime('%d-%m-%Y')
133         nowTime = now.strftime('%H:%M:%S')
134
135         try:
136             with open(filename, 'a') as csvfile:
137                 fileWriter = csv.writer(csvfile, delimiter=',')
138                 fileWriter.writerow([LName,fName,self.__moduleCode,self.__currentMark,nowDate,nowTime])
139         except IOError:
140             print('error')
141         else:
142             return
143
144
145     def exitTest(self):
146
147         return
148
149
150 #//\\//\\//\\//\\/#
151 #APPLICATION CLASS#
152 #//\\//\\//\\//\\/#
153
154 class Application(tk.Tk):
155
156     def __init__(self, *args, **kwargs):
157
158         tk.Tk.__init__(self, *args, **kwargs)
159
160         tk.Tk.wm_title(self, "Teams3's DQS Coursework Application")
161
162         self.__container = tk.Frame(self)
163         self.__container.grid()
164         self.__container.grid_rowconfigure(0, weight=1)
165         self.__container.grid_columnconfigure(0, weight=1)
166
167         self.createMenuBar(self.__container)
168
169         self.frames = {}
170
171         frame = LoginPage(self.__container, self)
172         self.frames[LoginPage] = frame
173         frame.grid(row=0, column=0, sticky="nsew")
174
175
176
177
178         self.show_frame(LoginPage)
179
180     def createMenuBar(self, container):
181         menubar = tk.Menu(container)
182         basicMenu = tk.Menu(menubar, tearoff=0)
183         basicMenu.add_command(label="Home", command=lambda: self.show_frame(HomePage))
184         basicMenu.add_separator()
185         basicMenu.add_command(label="Settings", command=lambda: popupMessage("Not Running yet"))
186         basicMenu.add_command(label="Your test scores", command=lambda:  self.show_frame(TestScores))
187         basicMenu.add_command(label="Logout", command=lambda: popupMessage("Not Running yet"))
188         basicMenu.add_separator()
189         basicMenu.add_command(label="Exit", command=quit)
190         infoMenu = tk.Menu(menubar,tearoff=0)
191         infoMenu.add_command(label="Help", command=lambda: popupMessage("Not Running yet"))
192         infoMenu.add_command(label="Feedback", command=lambda: self.show_frame(UserFeedback)) #New Line Here
193         infoMenu.add_separator()
194         infoMenu.add_command(label="About", command=lambda: popupMessage("Not Running yet"))
195         menubar.add_cascade(label="Menu", menu=basicMenu)
196         menubar.add_cascade(label="Help", menu=infoMenu)
197
198         tk.Tk.config(self, menu=menubar)
199
200     def validLogin(self):
201
202         for F in ( HomePage, TestScores, SearchPage):
203
204             frame = F(self.__container, self)
205             self.frames[F] = frame
206             frame.grid(row=0, column=0, sticky="nsew")
207
208         frame = TestModule001(self.__container, self, "001", 'An Introduction to HTML')
209         self.frames[TestModule001] = frame
210         frame.grid(row=0, column=0, sticky='nsew')
211
212         frame = TestModule002(self.__container, self, "002", 'HTML')
213         self.frames[TestModule002] = frame
214         frame.grid(row=0, column=0, sticky='nsew')
215
216         frame = LessonModule001(self.__container, self, "001")
217         self.frames[LessonModule001] = frame
218         frame.grid(row=0, column=0, sticky='nsew')
219
220         frame = LessonModule002(self.__container, self, "002")
221         self.frames[LessonModule002] = frame
```

```
222         frame.grid(row=0, column=0, sticky='nsew')
223 
224         frame = Editor001(self.__container, self, "001")
225         self.frames[Editor001] = frame
226         frame.grid(row=0, column=0, sticky='nsew')
227 
228         frame = Editor002(self.__container, self, "002")
229         self.frames[Editor002] = frame
230         frame.grid(row=0, column=0, sticky='nsew')
231 
232         frame = UserFeedback(self.__container, self, ['Module - 001', 'Module - 002'])
233         self.frames[UserFeedback] = frame
234         frame.grid(row=0, column=0, sticky='nsew')
235 
236         frame = ReviewFeedback(self.__container, self)
237         self.frames[ReviewFeedback] = frame
238         frame.grid(row=0, column=0, sticky='nsew')
239 
240 
241     def show_frame (self, cont):
242 
243         if cont != LoginPage:
244             self.createMenuBar(tk.Frame(self))
245         frame = self.frames[cont]
246         frame.tkraise()
247 
248 #//\\//\\//\\//\\/#
249 #MENU FRAME CLASS#
250 #//\\//\\//\\//\\/#
251 
252 class MenuFrame(tk.Frame):
253     'Menu Frame'
254 
255     def __init__(self, parent, controller):
256         'Initialise all widgets of the menu frame.'
257         tk.Frame.__init__(self, parent)
258 
259         # self.__column0xpad = 150
260         # self.__row0ypad = 75
261 
262         self.__column0xpad = 50
263         self.__row0ypad = 40
264 
265         self.lblTitle = tk.Label(self,bg='white', fg='#585858', text="Welcome to the Home Page", justify='center', font=EXTRA_LARGE_FONT, wraplength=500)
266         self.lblTitle.grid( row=0, column=0,  columnspan=1,  rowspan=1, padx=(self.__column0xpad,50), pady=(self.__row0ypad,75), sticky="n")
267 
268         self.lblMessage = tk.Label(self,bg='#DDDDDD', text="Welcome to Teams3's DQS Coursework Application, Use of Teams's DQS Courswork Programme is governed by Cardiff University Information Services Regulations and Policies. By logging in you agree to comply with these policies and regulations and you accept the use of cookies which the Blackboard software application puts on your computer, as well as how they collect, store and use data which you input.", justify='left', font=NORMAL_FONT, wraplength=400)
269         self.lblMessage.grid(row=1, column=0, columnspan=1, rowspan=1, padx=(self.__column0xpad,50), pady=(0,50), sticky="w")
270 
271         self.lblInstructions = tk.Label(self, text="Select a module, then click to start the lesson or start the test", justify="left", font=NORMAL_FONT, wraplength=300)
272         self.lblInstructions.grid(row=2, column=0, columnspan=1, rowspan=1, padx=(self.__column0xpad,0), pady=(75,0), sticky="w")
273 
274         self.lblModule = tk.Label(self, text='Module:' , justify='left', font=NORMAL_FONT, wraplength=100)
275         self.lblModule.grid(row=1, column=1, columnspan=1, rowspan=1, padx=(0,0), pady=(0,0), sticky="ne")
276 
277         self.listModule = tk.Listbox(self, height= 5, width=50, font=SMALL_FONT, selectmode=tk.SINGLE)
278         self.scroll = ttk.Scrollbar(self, command= self.listModule.yview)
279         self.listModule.configure(yscrollcommand=self.scroll.set)
280 
281         for item in ["001 - An Introduction to HTML", "002 - HTML"]:
282             self.listModule.insert(tk.END, item)
283 
284         self.listModule.selection_set(0, tk.END)
285         self.listModule.focus_set()
286 
287         self.listModule.grid(row=1, column=2, columnspan=1, rowspan=1, padx=(0,0), pady=(0,0), sticky="nw")
288         self.scroll.grid(row=1, column=3, columnspan=1,  rowspan=1, padx=(0,0), pady=(0,0), sticky="ns")
289         self.listModule.activate(0)
290 
291         self.butLesson = tk.Button(self, text="View Lesson", font=LARGE_BUTTON_FONT, height= 2, width=15, relief=tk.GROOVE,bg='white', fg='green', activebackground='black', activeforeground='green')
292         self.butLesson.bind("<Enter>", lambda event, x=self.butLesson: x.configure(bg="#80A0FF"))
293         self.butLesson.bind("<Leave>", lambda event, x=self.butLesson: x.configure(bg="#C5D0C8"))
294         self.butLesson.grid(row=2, column=1, columnspan=1, rowspan=1, padx=(0,0), pady=(100,0), sticky="w")
295 
296         self.butTest = tk.Button(self, text="Test", font=LARGE_BUTTON_FONT, height= 2, width=15, relief=tk.GROOVE, bg='white', fg='green', activebackground='black', activeforeground='red')
297         self.butTest.bind("<Enter>", lambda event, x=self.butTest: x.configure(bg="#80A0FF"))
298         self.butTest.bind("<Leave>", lambda event, x=self.butTest: x.configure(bg="#C5D0C8"))
299         self.butTest.grid(row=2, column=2, columnspan=1, rowspan=1, padx=(0,0), pady=(100,0), sticky="e")
300 
301         self.butEdit = tk.Button(self, text="Edit Lesson", font=LARGE_BUTTON_FONT, height= 2, width=15, relief=tk.GROOVE, bg='white', fg='green', activebackground='black', activeforeground='blue')
302         self.butEdit.bind("<Enter>", lambda event, x=self.butEdit: x.configure(bg="#80A0FF"))
303         self.butEdit.bind("<Leave>", lambda event, x=self.butEdit: x.configure(bg="#C5D0C8"))
304         self.butEdit.grid(row=3, column=2, columnspan=1, rowspan=1, padx=(0,0), pady=(50,0), sticky="e")
305 
306         self.butFeedback = tk.Button(self, text="View Feedback", font=LARGE_BUTTON_FONT, height= 2, width=15, relief=tk.GROOVE, bg='white', fg='green', activebackground='black', activeforeground='green')
307         self.butFeedback.bind("<Enter>", lambda event, x=self.butFeedback: x.configure(bg="#80A0FF"))
308         self.butFeedback.bind("<Leave>", lambda event, x=self.butFeedback: x.configure(bg="#C5D0C8"))
309         self.butFeedback.grid(row=3, column=1, columnspan=1, rowspan=1, padx=(0,0), pady=(50,0), sticky="e")
310 
311 #//\\//\\//\\//\\/#
312 #HOME  PAGE  CLASS#
313 #//\\//\\//\\//\\/#
314 
315 class HomePage (MenuFrame):
316 
317     def __init__(self, parent, controller):
318         MenuFrame.__init__(self, parent, controller)
319         self.butTest.configure(command=lambda: self.showSelectedModuleTest(controller))
320         self.butLesson.configure(command=lambda: self.showSelectedModuleLesson(controller))
321         self.butEdit.configure(command=lambda: self.showSelectedModuleEditor(controller))
322         self.butFeedback.configure(command=lambda: controller.show_frame(ReviewFeedback))
323 
324     def showSelectedModuleEditor(self, controller):
```

```
325
326         selection = self.listModule.curselection()
327         if (len(selection) == 1):
328             if (selection[0] == 0):
329                 controller.show_frame(Editor001)
330             elif (selection[0] == 1):
331                 controller.show_frame(Editor002)
332
333     def showSelectedModuleTest(self, controller):
334
335         selection = self.listModule.curselection()
336         if (len(selection) == 1):
337             if (selection[0] == 0):
338                 controller.show_frame(TestModule001)
339             elif (selection[0] == 1):
340                 controller.show_frame(TestModule002)
341
342     def showSelectedModuleLesson(self, controller):
343
344         selection = self.listModule.curselection()
345         if (len(selection) == 1):
346             if (selection[0] == 0):
347                 controller.show_frame(LessonModule001)
348             elif (selection[0] == 1):
349                 controller.show_frame(LessonModule002)
350
351 #//\\//\\//\\//\\//\\/#
352 #LECTURER FRAME CLASS#
353 #//\\//\\//\\//\\//\\/#
354
355 class LectureFrame(tk.Frame):
356
357     def __init__(self, parent , controller, moduleCode):
358         tk.Frame.__init__(self, parent)
359         self.__lesson = "lesson_"+ moduleCode+ ".txt"
360         self.__title = 'title'
361         #self.grid()
362         self.draw_widgets()
363         self.button()
364
365
366     def draw_widgets(self):
367
368         self.cframe = tk.Frame(self)
369         self.cframe.grid(row=0, column=0, padx=(100,0), sticky='news')
370         self.canv = tk.Canvas(self.cframe, width=976, height=560)  #612                    #this wraps text widgets within
371         self.canv.grid(row=0, column=0, sticky='news')                          #a canvas so that they can be
372         self.vscroll = tk.Scrollbar(self.cframe, orient=tk.VERTICAL, command=self.canv.yview) #scrolled through
373         self.vscroll.grid(row=0, column=1, sticky='ns')
374         self.canv["yscrollcommand"] = self.vscroll.set
375         self.aframe = tk.Frame(self.canv)
376         id = self.canv.create_window(0,0,window=self.aframe, anchor='nw')
377
378         Num = 0
379         file = open(self.__lesson,"r")
380         for line in file:
381             if line in ['\n', '\r\n']: #if line is blank move down a row
382                 Num += 1
383             self.fileInputFrame = tk.Frame(self.aframe)
384             self.fileInputFrame.grid(row = Num)
385             count = 0
386             wordLen = 0
387             for word in line.split():
388                 wordLen = wordLen + len(word)
389                 if wordLen > 96:
390                     Num += 1
391                     self.fileInputFrame = tk.Frame(self.aframe) #is the length exceeds 110 move down a row
392                     self.fileInputFrame.grid(row = Num)
393                 if word[0] == ".":
394                     if word[1] == "B":
395                         self.w = tk.Label(self.fileInputFrame, text = word[2:], font = ("Times", 15, "bold"))
396                         self.w.grid(row = 0, column = count)
397                         count += 1
398                     elif word[1] == "U":
399                         self.w = tk.Label(self.fileInputFrame, text = "\u2286", font = ("Times", 10, "bold"))
400                         self.w.grid(row = 0, column = count)
401                         count += 1
402                     elif word[1] == "I":
403                         self.w = tk.Label(self.fileInputFrame, text = word[2:], font = ("Times", 15, "italic"))
404                         self.w.grid(row = 0, column = count)
405                         count += 1
406                     elif word[1] == "F":
407                         if "/" in word:
408                             word = word[2:].split("/")
409                             if len(word[0]) > len(word[1]):
410                                 Avg = len(word[0])
411                             else:
412                                 Avg = len(word[1])  #gernerate an undeline length based on the longest of the 2
413                             underline = ""          #items in the fraction
414                             for i in range(0,Avg+5):
415                                 underline = underline + "-"
416                             self.SubFrame = tk.Frame(self.fileInputFrame)
417                             self.SubFrame.grid(row = 0,column = count)
418                             self.w = tk.Label(self.SubFrame, text = word[0], font = ("Times", 12), anchor = 's')
419                             self.w.grid(row = 0, column = 0) #top of fraction
420                             self.w = tk.Label(self.SubFrame, text = underline, font = ("Times", 4))
421                             self.w.grid(row = 1, column = 0) #underline of fraction
422                             self.v = tk.Label(self.SubFrame, text = word[1], font = ("Times", 12), anchor = 'n')
423                             self.v.grid(row = 2, column = 0) #bottom of fraction
424                             count += 1
425                     elif word[1] == "S":
426                         if word[2] == "P":
427                             self.SubFrame = tk.Frame(self.fileInputFrame)
428                             self.SubFrame.grid(row = 0,column = count)
429                             self.w = tk.Label(self.SubFrame, text = word[3:], font = ("Times", 9))
430                             self.w.grid(row = 0, column = 0)
431                             self.v = tk.Label(self.SubFrame, text = "", font = ("Times", 6))
432                             self.v.grid(row = 1, column = 0) #superscript
433                             count += 1
434                         elif word[2] == "B":
435                             self.SubFrame = tk.Frame(self.fileInputFrame)
```

```
                            self.SubFrame.grid(row = 0,column = count)
                            self.w = tk.Label(self.SubFrame, text = word[3:], font = ("Times", 9))
                            self.w.grid(row = 1, column = 0)
                            self.v = tk.Label(self.SubFrame, text = "", font = ("Times", 6))
                            self.v.grid(row = 0, column = 0) #subscript
                            count += 1
                    else:
                        self.w = tk.Label(self.fileInputFrame, text = word, font = ("Times", 15))
                        self.w.grid(row = 0, column = count)
                        count += 1 #still print text if it starts with . and is not a command
                else:
                    self.w = tk.Label(self.fileInputFrame, text = word, font = ("Times", 15))
                    self.w.grid(row = 0, column = count)
                    count += 1
            Num += 1
            self.aframe.update_idletasks()
            self.canv["scrollregion"]=self.canv.bbox(tk.ALL) #update scroll region

    def button(self):
        self.ButtonFrame = tk.Frame(self)
        self.ButtonFrame.grid(row = 3)


        self.butBack = ttk.Button(self.ButtonFrame, text='Go Back')
        #butEdit['command'] = self.Back
        self.butBack.grid(row = 0,column = 0,  pady=(25,0), padx=(0,50))

        self.butTest = ttk.Button(self.ButtonFrame, text='Take Test')
        #butEdit['command'] = self.Test
        self.butTest.grid(row = 0,column = 2, pady=(25,0) )

    def setCommands(self, controller, menu, test):

        self.butBack.configure(command=lambda: controller.show_frame(menu))
        self.butTest.configure(command=lambda: controller.show_frame(test))

#//\\//\\//\\//\\//#
#EDITOR FRAME CLASS#
#//\\//\\//\\//\\//#

class EditorFrame(tk.Frame):
    def __init__(self, parent , controller, moduleCode):
        tk.Frame.__init__(self, parent)
        self.__lesson = "lesson_"+ moduleCode+ ".txt"
        #root.protocol('WM_DELETE_WINDOW', self.exit) ,this will ask if you want to save when you press the red [X]
        self.textbox()
        self.menubar()

    def textbox(self):
        self.textPad = scrolledtext.ScrolledText(self, width = 115, height = 32)
        read = open(self.__lesson,"r")
        self.textPad.insert('1.0',read.read()) #creates a textbox with the contents of the file
        self.textPad.grid(column = 0, row = 1, padx=(50,0))

    def save(self):
        file = open(self.__lesson,"w")
        data = self.textPad.get('1.0', tk.END) #reads the lines and saves them to the text file
        file.write(data)
        file.close()

    def exit(self, controller, menu):
        if tm.askyesno("","Do you wish to save before exit?"):
            self.save()
        controller.show_frame(menu)


    def help(self):
        label = tm.showinfo("Help", "Basic commands: .B - bold .I - italics .SB - subscript .SP - superscript .F - fraction .U - subset")

    def menubar(self):
        self.ButtonFrame = tk.Frame(self)
        self.ButtonFrame.grid(row = 0)

        self.butSave = ttk.Button(self.ButtonFrame, text='Save')
        self.butSave['command'] = self.save
        self.butSave.grid(row = 0,column = 1, pady=(30,30), padx=(35,35))

        self.butHelp = ttk.Button(self.ButtonFrame, text='Help')
        self.butHelp['command'] = self.help
        self.butHelp.grid(row = 0,column = 0, pady=(30,30))

        self.butBack = ttk.Button(self.ButtonFrame, text='Back')
        self.butBack.grid(row = 0,column = 2, pady=(30,30))

    def setCommands(self, controller, menu):
        self.butBack.configure(command=lambda: self.exit(controller, menu))

#//\\//\\//\\//\\//\\//\\//\\//#
#SEARCH SCORE FRAME CLASS#
#//\\//\\//\\//\\//\\//\\//\\#

class SearchScoresFrame(tk.Frame):

    def __init__(self, parent , controller):

        tk.Frame.__init__(self, parent)
        label = tk.Label(self, text="Test Results",font = L_FONT)

        mbutton1 = tk.Button(self,text= "Display all scores",command = lambda: controller.show_frame(testScores) ,font = L_FONT)
        mbutton1.pack(pady=5)


        userEnt = tk.Entry(self)
        userEnt.pack(pady=5)
        userEnt.get()

        def printer():

            userSearch = userEnt.get()
            print( str(userSearch) )

```

```
547             cr = csv.reader(open("test_marks.csv"))
548 
549             for row in cr:
550                 if userSearch in row:
551                     text = tk.Label(self, text = "   ".join(row),font = L_FONT )
552                     text.pack()
553 
554         tk.Button(self, text='Search', command=printer).pack()
555 
556         label2 = tk.Label(self, text="Surname-Forename-Module-Mark-Test date-Test time",font = L_FONT)
557         label.pack(pady= 10)
558         label2.pack(pady= 13)
559 
560 #//\\//\\//\\//\\//\\//\#
561 #TEST SCORE FRAME CLASS#
562 #//\\//\\//\\//\\//\\//#
563 
564 class TestScoresFrame(tk.Frame):
565 
566     def __init__(self, parent, controller):
567         tk.Frame.__init__(self, parent)
568 
569         self.mbutton1 = tk.Button(self,text= "Seach Page",font = L_FONT)
570         self.mbutton1.pack(pady=5)
571 
572         label = tk.Label(self, text="All Test Results",font = L_FONT).pack()
573         label2 = tk.Label(self, text="Surname-Forename-Module-Mark-Test date-Test time",font = L_FONT)
574         label2.pack(pady= 13)
575 
576         cro = csv.reader(open("test_marks.csv"))
577         for row in cro:
578             if "001" in row:
579                 text1 = tk.Label(self, text ="   ".join(row), font= L_FONT).pack()
580 
581     def setCommands(self, controller, searchPage):
582         self.mbutton1.configure(command = lambda: controller.show_frame(searchPage))
583 
584 #//\\//\\//\\///\#
585 #TEST FRAME CLASS#
586 #//\\//\\\\//\\#
587 
588 class  TestFrame(tk.Frame):
589     'Test frame'
590 
591     def __init__(self, parent, controller, mCode, mName, lCompleted):
592         'Initialise all main widgets of the test frame.'
593         tk.Frame.__init__(self, parent)
594         self.__associatedModule = Module(mCode, mName, lCompleted)
595 
596         # self.__column0xpad = 150
597         # self.__row0ypad = 75
598 
599         self.__column0xpad = 50
600         self.__row0ypad = 40
601 
602         self.setIntructionPage()
603 
604         self.questionTemplate()
605         self.hideQuestionTemplate()
606 
607         self.feedbacktemplate()
608         self.hideFeedbackTemplate()
609 
610     def setIntructionPage(self):
611         'Create widgets for the instruction for the test.'
612 
613         #styleEdit = ttk.Style()
614         #styleEdit.configure('TButton', width=15, font=LARGE_BUTTON_FONT)
615 
616         self.titleVariable = tk.StringVar()
617         self.titleVariable.set(self.__associatedModule.getModuleName() + " Test Instructions")
618 
619         self.labelVariable = tk.StringVar()
620         self.labelVariable.set("You have selected to do the test for " + self.__associatedModule.getModuleCode() + " you only have one attempt. Please read the
    instructions thoroughly before you start the test.")
621 
622         self.lblTitle = tk.Label(self, textvariable=self.titleVariable, justify='center', font=EXTRA_LARGE_FONT, wraplength=400)
623         self.lblTitle.grid( row=0, column=0,  columnspan=1,  rowspan=1, padx=(self.__column0xpad,50), pady=(self.__row0ypad,75), sticky="n")
624 
625         self.lblMessage = tk.Label(self, textvariable=self.labelVariable, justify='right', font=NORMAL_FONT, wraplength=400)
626         self.lblMessage.grid(row=1, column=0, columnspan=1, rowspan=1, padx=(self.__column0xpad,50), pady=(0,50), sticky="w")
627 
628 
629         self.butMenu = tk.Button(self, text="Back to Menu", font=LARGE_BUTTON_FONT, height= 2, width=15, relief=tk.GROOVE, bg="#C5D0C8")
630         self.butMenu.bind("<Enter>", lambda event, x=self.butMenu: x.configure(bg="#80A0FF"))
631         self.butMenu.bind("<Leave>", lambda event, x=self.butMenu: x.configure(bg="#C5D0C8"))
632         self.butMenu.grid(row=2, column=2, columnspan=1, rowspan=1, padx=(0,0), pady=(100,0), sticky="e")
633 
634         self.butStart = tk.Button(self, text="Start Test", font=LARGE_BUTTON_FONT, height= 2, width=15, relief=tk.GROOVE, bg="#C5D0C8")
635         self.butStart.bind("<Enter>", lambda event, x=self.butStart: x.configure(bg="#80A0FF"))
636         self.butStart.bind("<Leave>", lambda event, x=self.butStart: x.configure(bg="#C5D0C8"))
637         self.butStart.grid(row=2, column=3, columnspan=1, rowspan=1, padx=(150,0), pady=(100,0), sticky="w")
638 
639     def questionTemplate(self):
640         'Creates widgets for the basic template for the test questions.'
641 
642         self.lblQuestionNumber = tk.Label(self, text='', justify='left', font=EXTRA_LARGE_FONT, wraplength=300 )
643         self.lblQuestionNumber.grid(row=0, column=0, columnspan=1, rowspan=1, padx=(self.__column0xpad, 50), pady=(self.__row0ypad,10), sticky='nw')
644 
645         self.lblQuestion = tk.Label(self, text='', justify='left', font=NORMAL_FONT, wraplength=600, height=6, width=100)
646         self.lblQuestion.grid(row=1, column=0, columnspan=3, rowspan=1, padx=(0,0), pady=(0,10), sticky='nw')
647 
648         self.lblAnswerSelect = tk.Label(self, text='Select Answer:', justify='left', font=LARGE_FONT, wraplength=200)
649         self.lblAnswerSelect.grid(row=2, column=0, columnspan=1, rowspan=1, padx=(self.__column0xpad, 50), pady=(0,20), sticky='nw')
650 
651         self.butNext = tk.Button(self, text="Next", font=LARGE_BUTTON_FONT, height= 2, width=15, relief=tk.GROOVE, bg="#C5D0C8")
652         self.butNext.bind("<Enter>", lambda event, x=self.butNext: x.configure(bg="#80A0FF"))
653         self.butNext.bind("<Leave>", lambda event, x=self.butNext: x.configure(bg="#C5D0C8"))
654         self.butNext.grid(row=5, column=2, columnspan=1, rowspan=1, padx=(0,0), pady=(100,0), sticky="es")
655         self.butNext.configure(command=lambda: self.nextQuestion())
656 
```

```
657         self.createRadioButtons()
658 
659     def createRadioButtons(self):
660         'Creates the radio button widgets used to select answers.'
661 
662         styleEdit = ttk.Style()
663         styleEdit.configure('TRadiobutton', font=NORMAL_FONT)
664 
665         self.__varAnswer = tk.StringVar()
666 
667         self.radbutAnswer1 = ttk.Radiobutton(self, text='1', variable=self.__varAnswer, value='1')
668         self.radbutAnswer2 = ttk.Radiobutton(self, text='2', variable=self.__varAnswer, value='2')
669         self.radbutAnswer3 = ttk.Radiobutton(self, text='3', variable=self.__varAnswer, value='3')
670         self.radbutAnswer4 = ttk.Radiobutton(self, text='4', variable=self.__varAnswer, value='4')
671 
672         self.radbutAnswer1.grid(row=3, column=0, columnspan=1, rowspan=1, padx=(self.__column0xpad,0), pady=(0,0), sticky="nw")
673         self.radbutAnswer2.grid(row=4, column=0, columnspan=1, rowspan=1, padx=(self.__column0xpad,0), pady=(15,0), sticky="nw")
674         self.radbutAnswer3.grid(row=3, column=1, columnspan=1, rowspan=1, padx=(50,0), pady=(15,0), sticky="nw")
675         self.radbutAnswer4.grid(row=4, column=1, columnspan=1, rowspan=1, padx=(50,0), pady=(15,0), sticky="nw")
676 
677     def feedbacktemplate(self):
678 
679         self.varFeedbackTitle = tk.StringVar()
680         self.varFeedbackTitle.set(self.__associatedModule.getModuleName() + " Test Feedback")
681 
682         self.varFeedback = tk.StringVar()
683         self.varFeedback.set("Thank you for completing the test for the module " + self.__associatedModule.getModuleCode() + ". Please review each question and
    look at the ones that you got wrong, if any.")
684 
685         self.lblFeedbackTitle = tk.Label(self, textvariable=self.varFeedbackTitle, justify='center', font=EXTRA_LARGE_FONT, wraplength=400)
686         self.lblFeedbackTitle.grid( row=0, column=0,  columnspan=1,  rowspan=1, padx=(self.__column0xpad,50), pady=(self.__row0ypad,20), sticky="n")
687 
688         self.lblFeedbackMessage = tk.Label(self, textvariable=self.varFeedback, justify='left', font=NORMAL_FONT, wraplength=400)
689         self.lblFeedbackMessage.grid(row=1, column=0, columnspan=1, rowspan=1, padx=(self.__column0xpad,50), pady=(20,50), sticky="w")
690 
691         self.lblMarks = tk.Label(self, text='0/20', justify='left', font=EXTRA_LARGE_FONT)
692         self.lblMarks.grid(row=0, column=2, columnspan=1, rowspan=1, padx=(400,0), pady=(self.__row0ypad,20), sticky="w")
693 
694         self.addListbox()
695 
696         self.addDetailsTemplate()
697 
698         self.butFeedbackMenu = tk.Button(self, text="Finished", font=LARGE_BUTTON_FONT, height= 2, width=15, relief=tk.GROOVE, bg="#C5D0C8")
699         self.butFeedbackMenu.bind("<Enter>", lambda event, x=self.butFeedbackMenu: x.configure(bg="#80A0FF"))
700         self.butFeedbackMenu.bind("<Leave>", lambda event, x=self.butFeedbackMenu: x.configure(bg="#C5D0C8"))
701         self.butFeedbackMenu.grid(row=6, column=2, columnspan=1, rowspan=1, padx=(300,0), pady=(75,0), sticky="w")
702 
703     def addDetailsTemplate(self):
704 
705         self.lblFeedbackQuestion = tk.Label(self, text='', justify='left', font=NORMAL_FONT, wraplength=500, height=5)
706         self.lblFeedbackQuestion.grid(row=1, column=2, columnspan=2, rowspan=1, padx=(40,50), pady=(20,20), sticky="w")
707 
708         self.lblCorrectTitle = tk.Label(self, text='Correct Answer:', font=NORMAL_FONT, justify='left')
709         self.lblCorrectTitle.grid(row=2, column=2, columnspan=1, rowspan=1, padx=(40,0), pady=(25,0), sticky='w')
710 
711         self.lblCorrect = tk.Label(self, text='', font=NORMAL_FONT, justify='left')
712         self.lblCorrect.grid(row=3, column=2, columnspan=1, rowspan=1, padx=(40,0), pady=(0,0), sticky='w')
713 
714         self.lblGivenTitle = tk.Label(self, text='Selected Answer:', font=NORMAL_FONT, justify='left')
715         self.lblGivenTitle.grid(row=4, column=2, columnspan=1, rowspan=1, padx=(40,0), pady=(25,0), sticky='w')
716 
717         self.lblGiven = tk.Label(self, text='', font=NORMAL_FONT, justify='left')
718         self.lblGiven.grid(row=5, column=2, columnspan=1, rowspan=1, padx=(40,0), pady=(0,0), sticky='w')
719 
720     def addListbox(self ):
721 
722         self.listQuestions = tk.Listbox(self, height= 5, width=50, font=NORMAL_FONT, selectmode=tk.SINGLE)
723         self.scroll = ttk.Scrollbar(self)
724 
725         self.listQuestions.configure(yscrollcommand=self.scroll.set)
726         self.scroll.configure( command= self.listQuestions.yview)
727 
728         self.listQuestions.selection_set(0, tk.END)
729         self.listQuestions.focus_set()
730 
731         self.listQuestions.grid(row=2, column=0, columnspan=1, rowspan=3, padx=(self.__column0xpad,0), pady=(25,0), sticky="nw")
732         self.scroll.grid(row=2, column=1, columnspan=1,  rowspan=3, padx=(0,0), pady=(25,0), sticky="ns")
733 
734         self.listQuestions.bind("<<ListboxSelect>>", self.listQuestionClick)
735 
736         pass
737 
738     def listQuestionClick(self, event):
739 
740         selected = self.listQuestions.curselection()
741         questionNumber = selected[0] + 1
742         test = self.__associatedModule.getModuleTest()
743         self.displayTestQuestionForFeedback(test, questionNumber)
744 
745     def displayQuestionDetails(self, question):
746         'Edits widgets of the created in questionTemplate() with the details of the question passes as an argument'
747 
748         self.lblQuestionNumber.configure(text='Question ' + str(question.getQuestionNumber()))
749         self.lblQuestion.configure(text=str(question.getQuestionInfo()))
750 
751         answerList =[]
752         answerList.append(question.getCorrectAnswer())
753         for answer in question.getIncorrectAnswers():
754             answerList.append(answer)
755 
756         self.__varAnswer.set('')
757 
758         answer = rdm.choice(answerList)
759         self.radbutAnswer1.configure(text=answer, value=answer)
760         answerList.remove(answer)
761 
762         answer = rdm.choice(answerList)
763         self.radbutAnswer2.configure(text=answer, value=answer)
764         answerList.remove(answer)
765 
766         answer = rdm.choice(answerList)
```

```
767         self.radbutAnswer3.configure(text=answer, value=answer)
768         answerList.remove(answer)
769 
770         answer = rdm.choice(answerList)
771         self.radbutAnswer4.configure(text=answer, value=answer)
772         answerList.remove(answer)
773 
774     def questionFeedback(self, question, givenAnswer, correct):
775         'Method used to display question feedback.'
776 
777         popupFeedback = tk.Tk()
778         popupFeedback.geometry("500x400+500+200")
779         popupFeedback.grid()
780         popupFeedback.wm_title("Instant feedback!")
781 
782         lblTitle = tk.Label(popupFeedback, text='Instant feedback', font=LARGE_FONT, justify='center')
783         lblTitle.grid(row=0, column=0, columnspan=1, rowspan=1, padx=(25,0), pady=(25,0), sticky="n")
784 
785         lblQuestionTitle = tk.Label(popupFeedback, text='Question:', font=NORMAL_FONT, justify='left')
786         lblQuestionTitle.grid(row=1, column=0, columnspan=1, rowspan=1, padx=(25,0), pady=(25,0), sticky='w')
787 
788         lblQuestion = tk.Label(popupFeedback, text=question.getQuestionInfo(), font=NORMAL_FONT, justify='left', wraplength=450)
789         lblQuestion.grid(row=2, column=0, columnspan=1, rowspan=1, padx=(25,0), pady=(0,0), sticky='w')
790 
791         lblCorrectTitle = tk.Label(popupFeedback, text='Correct Answer:', font=NORMAL_FONT, justify='left')
792         lblCorrectTitle.grid(row=3, column=0, columnspan=1, rowspan=1, padx=(25,0), pady=(25,0), sticky='w')
793 
794         lblCorrect = tk.Label(popupFeedback, text=question.getCorrectAnswer(), font=NORMAL_FONT, justify='left')
795         lblCorrect.grid(row=4, column=0, columnspan=1, rowspan=1, padx=(25,0), pady=(0,0), sticky='w')
796 
797         lblGivenTitle = tk.Label(popupFeedback, text='Selected Answer:', font=NORMAL_FONT, justify='left')
798         lblGivenTitle.grid(row=5, column=0, columnspan=1, rowspan=1, padx=(25,0), pady=(25,0), sticky='w')
799 
800         lblGiven = tk.Label(popupFeedback, text=givenAnswer, font=NORMAL_FONT, justify='left')
801         lblGiven.grid(row=6, column=0, columnspan=1, rowspan=1, padx=(25,0), pady=(0,0), sticky='w')
802 
803         if correct:
804             lblGiven.configure(fg='blue')
805         else:
806             lblGiven.configure(fg='red')
807 
808         butOkay = ttk.Button(popupFeedback, text="Okay", command = popupFeedback.destroy)
809         butOkay.grid(row=7, column=0, columnspan=1, rowspan=1, padx=(0,0), pady=(25,0), sticky="s")
810         butOkay.focus_set()
811 
812     def promptAnswer(self):
813         'Method used to display question feedback.'
814 
815         popupFeedback = tk.Tk()
816         popupFeedback.geometry("300x100+700+400")
817 
818         popupFeedback.wm_title("Attention!")
819         labTitle = tk.Label(popupFeedback, text='You must select an answer\n before you can continue.', font=NORMAL_FONT)
820         labTitle.pack(side="top", pady=10)
821         butOkay = ttk.Button(popupFeedback, text="Okay", command = popupFeedback.destroy)
822         butOkay.pack()
823 
824     def markQuestion(self, test, givenAnswer):
825         'Mark the current question with the answer given and correct answer.'
826         correct = test.checkAnswer(givenAnswer)
827         if correct:
828             test.incCurrentMark()
829         question = test.getQuestionDetails()
830         self.questionFeedback(question, givenAnswer, correct)
831 
832     def displayTestQuestionForFeedback(self, test, questionNumber):
833         question = test.getQuestionDetails(questionNumber)
834         self.lblFeedbackQuestion.configure(text=question.getQuestionInfo())
835 
836         correct = question.getCorrectAnswer()
837         selected = test.getSelectedAnswer(questionNumber)
838 
839         self.lblCorrect.configure(text=correct)
840         self.lblGiven.configure(text=selected)
841 
842         if selected == correct:
843             self.lblGiven.configure(fg='green')
844         else:
845             self.lblGiven.configure(fg='red')
846 
847     def colourListBox (self, test):
848 
849         for index in range(0, test.getNumberOfQuestions()):
850             correct = test.getQuestionDetails(index + 1).getCorrectAnswer()
851             selected = test.getSelectedAnswer(index + 1)
852 
853             if selected == correct:
854                 self.listQuestions.itemconfig(index, fg='blue')
855             else:
856                 self.listQuestions.itemconfig(index,fg='red')
857 
858     def displayTestData(self, test):
859 
860         if self.listQuestions.size() != 0:
861             self.listQuestions.delete(0,self.listQuestions.size() -1 )
862         testLength = test.getNumberOfQuestions()
863 
864         for item in range(1,testLength + 1):
865             self.listQuestions.insert(tk.END, 'Question ' +str(item)  )
866 
867         self.lblMarks.configure(text=str(test.getCurrentMark()) + '/' + str(test.getNumberOfQuestions()))
868 
869         self.colourListBox(test)
870 
871         self.displayTestQuestionForFeedback(test, 1)
872         self.listQuestions.activate(0)
873         self.listQuestions.index(0)
874 
875     def nextQuestion(self):
876         'Method used when going to the next question.'
877         givenAnswer = self.__varAnswer.get()
```

```
878         test = self.__associatedModule.getModuleTest()
879         if givenAnswer == '':
880             self.promptAnswer()
881         else:
882             self.markQuestion(test, givenAnswer)
883             test.addSelectedAnswer(givenAnswer)
884             if test.getCurrentQuestionNumber() == test.getNumberOfQuestions() :
885                 test.saveMarks('ForenameTest','SurnameTest')
886                 self.hideQuestionTemplate()
887                 self.displayTestData(test)
888                 self.showFeedbackTemplate()
889             else:
890                 test.incCurrentQuestion()
891                 nextQuestionDetails = test.getQuestionDetails()
892                 self.displayQuestionDetails(nextQuestionDetails)
893 
894     def startTest (self):
895         'Method used to start the test.'
896         self.hideInstructions()
897 
898         startingQuestion = self.__associatedModule.getModuleTest().getQuestionDetails()
899 
900         self.displayQuestionDetails(startingQuestion)
901 
902         self.showQuestionTemplate()
903 
904     def showInstructions(self):
905         'Shows the instruction page widgets.'
906 
907         self.lblTitle.grid()
908         self.lblMessage.grid()
909         self.butMenu.grid()
910         self.butStart.grid()
911 
912     def showQuestionTemplate(self):
913         'Shows the question template widgets'
914 
915         self.lblQuestionNumber.grid()
916         self.lblQuestion.grid()
917         self.butNext.grid()
918         self.lblAnswerSelect.grid()
919         self.radbutAnswer1.grid()
920         self.radbutAnswer2.grid()
921         self.radbutAnswer3.grid()
922         self.radbutAnswer4.grid()
923 
924     def showFeedbackTemplate(self):
925 
926         self.lblFeedbackTitle.grid()
927         self.lblFeedbackMessage.grid()
928         self.lblMarks.grid()
929         self.butFeedbackMenu.grid()
930         self.listQuestions.grid()
931         self.scroll.grid()
932         self.lblFeedbackQuestion.grid()
933         self.lblCorrectTitle.grid()
934         self.lblCorrect.grid()
935         self.lblGivenTitle.grid()
936         self.lblGiven.grid()
937 
938     def hideInstructions(self):
939         'Hides the instruction page widgets'
940 
941         self.lblTitle.grid_remove()
942         self.lblMessage.grid_remove()
943         self.butMenu.grid_remove()
944         self.butStart.grid_remove()
945 
946     def hideQuestionTemplate(self):
947         'Hides the question template widgets'
948 
949         self.lblQuestionNumber.grid_remove()
950         self.lblQuestion.grid_remove()
951         self.butNext.grid_remove()
952         self.lblAnswerSelect.grid_remove()
953         self.radbutAnswer1.grid_remove()
954         self.radbutAnswer2.grid_remove()
955         self.radbutAnswer3.grid_remove()
956         self.radbutAnswer4.grid_remove()
957 
958     def hideFeedbackTemplate(self):
959 
960         self.lblFeedbackTitle.grid_remove()
961         self.lblFeedbackMessage.grid_remove()
962         self.lblMarks.grid_remove()
963         self.butFeedbackMenu.grid_remove()
964         self.listQuestions.grid_remove()
965         self.scroll.grid_remove()
966         self.lblFeedbackQuestion.grid_remove()
967         self.lblCorrectTitle.grid_remove()
968         self.lblCorrect.grid_remove()
969         self.lblGivenTitle.grid_remove()
970         self.lblGiven.grid_remove()
971 
972     def setCommands(self, controller, menu):
973 
974         self.butMenu.configure(command=lambda: controller.show_frame(menu))
975         self.butStart.configure(command=lambda: self.startTest())
976         self.butFeedbackMenu.configure(command=lambda: controller.show_frame(menu))
977 
978 class LessonModule001 (LectureFrame):
979 
980     def __init__(self, parent, controller, mCode):
981         LectureFrame.__init__(self, parent, controller, mCode)
982         self.setCommands(controller, HomePage, TestModule001)
983 
984 #//\\//\\//\\//\\//\\//\\#
985 #LESSON MODULE 002  CLASS#
986 #//\\//\\//\\//\\//\\//\\#
987 
988 class LessonModule002 (LectureFrame):
```

```
989
990     def __init__(self, parent, controller, mCode):
991         LectureFrame.__init__(self, parent, controller, mCode)
992         self.setCommands(controller, HomePage, TestModule002)
993
994
995 #//\\//\\//\\//\\//\\//\\//\\#
996 #FEED BACK SUBMIT FRAME CLASS#
997 #//\\//\\//\\//\\//\\//\\//\\#
998
999 class FeedbackSubmitFrame(tk.Frame):
1000
1001     def __init__(self, parent, controller, moduleList):
1002         tk.Frame.__init__(self, parent)
1003
1004         self.__column0xpad = 90
1005         self.__row0ypad = 50
1006
1007         self.lblTitle = tk.Label(self, text='User Feedback', justify='center', font=EXTRA_LARGE_FONT, wraplength=400)
1008         self.lblTitle.grid( row=0, column=0,  columnspan=2,  rowspan=1, padx=(self.__column0xpad,50), pady=(self.__row0ypad,0), sticky="n")
1009
1010         self.lblMessage = tk.Label(self, text='Your feeback on this application would be appreciated' , justify='left', font=NORMAL_FONT, wraplength=400)
1011         self.lblMessage.grid(row=1, column=0, columnspan=2, rowspan=1, padx=(self.__column0xpad,50), pady=(50,20),sticky="sw")
1012
1013
1014         self.butMenu = tk.Button(self, text="Back to Menu", font=LARGE_BUTTON_FONT, height= 2, width=15, relief=tk.GROOVE, bg="#C5D0C8")
1015         self.butMenu.bind("<Enter>", lambda event, x=self.butMenu: x.configure(bg="#80A0FF"))
1016         self.butMenu.bind("<Leave>", lambda event, x=self.butMenu: x.configure(bg="#C5D0C8"))
1017         self.butMenu.grid(row=8, column=0, columnspan=2, rowspan=1, padx=(self.__column0xpad,50), pady=(30,0), sticky="s")
1018
1019         self.butSubmit = tk.Button(self, text="Submit", font=LARGE_BUTTON_FONT, height= 2, width=15, relief=tk.GROOVE, bg="#C5D0C8")
1020         self.butSubmit.bind("<Enter>", lambda event, x=self.butSubmit: x.configure(bg="#80A0FF"))
1021         self.butSubmit.bind("<Leave>", lambda event, x=self.butSubmit: x.configure(bg="#C5D0C8"))
1022         self.butSubmit.grid(row=8, column=1, columnspan=3, rowspan=1, padx=(200,0), pady=(30,0), sticky="s")
1023         self.butSubmit.configure(command=lambda: self.submitFeedback())
1024
1025         self.addAnonymous()
1026         self.addNameOption()
1027         self.addListbox(moduleList)
1028         self.addFeedbackTextBox()
1029
1030     def addAnonymous(self):
1031         self.ckbxAnonymousvar = tk.IntVar()
1032         self.ckbxAnonymous = tk.Checkbutton(self, text='Select this box if you wish to remain anonymous', justify='left', font=NORMAL_FONT,
    variable=self.ckbxAnonymousvar)
1033         self.ckbxAnonymous.grid(row=2, column=0, columnspan=2, rowspan=1, padx=(self.__column0xpad,0), pady=(50,0), sticky="sw")
1034         self.ckbxAnonymous.configure(command=lambda:  self.checkboxClick())
1035
1036     def checkboxClick(self):
1037         if self.ckbxAnonymousvar.get() == 1:
1038             self.entNumber.configure(state=tk.DISABLED)
1039             self.entName.configure(state=tk.DISABLED)
1040         else:
1041             self.entNumber.configure(state=tk.NORMAL)
1042             self.entName.configure(state=tk.NORMAL)
1043
1044     def addNameOption(self):
1045
1046         self.lblName = tk.Label(self, text="Name", justify='left', font=NORMAL_FONT, wraplength=400)
1047         self.lblName.grid(row=3, column=0, columnspan=1, rowspan=1, padx=(self.__column0xpad,20), pady=(20,0), sticky="e")
1048
1049         self.entName = ttk.Entry(self, width=30)
1050         self.entName.grid(row=3, column=1, columnspan=1, rowspan=1, padx=(0,0), pady=(20,0), sticky="w")
1051
1052         self.lblNumber = tk.Label(self, text="Student Number", justify='left', font=NORMAL_FONT, wraplength=400)
1053         self.lblNumber.grid(row=4, column=0, columnspan=1, rowspan=1, padx=(self.__column0xpad,20), pady=(10,0), sticky="e")
1054
1055         self.entNumber = ttk.Entry(self, width=30)
1056         self.entNumber.grid(row=4, column=1, columnspan=1, rowspan=1, padx=(0,0), pady=(10,0), sticky="w")
1057
1058     def addListbox(self, moduleList):
1059
1060         self.listModulesLabel = tk.Label(self, text='Select module associated with your feedback or whether it is a general comment' , justify='left',
    font=NORMAL_FONT, wraplength=400)
1061         self.listModulesLabel.grid(row=1, column=2, columnspan=2, rowspan=1, padx=(0,0), pady=(0,20), sticky="sw")
1062
1063         self.listModules = tk.Listbox(self, height= 3, width=60, font=SMALL_FONT, selectmode=tk.SINGLE)
1064         self.scroll = ttk.Scrollbar(self)
1065
1066         self.listModules.configure(yscrollcommand=self.scroll.set)
1067         self.scroll.configure( command= self.listModules.yview)
1068
1069         self.listModules.selection_set(0, tk.END)
1070         self.listModules.focus_set()
1071
1072         self.listModules.grid(row=2, column=2, columnspan=2, rowspan=1, padx=(0,0), pady=(20,0), sticky="nw")
1073         self.scroll.grid(row=2, column=4, columnspan=1,  rowspan=1, padx=(0,0), pady=(20,0), sticky="ns")
1074
1075         for mod in moduleList:
1076             self.listModules.insert(tk.END, mod)
1077         self.listModules.insert(tk.END, 'General Comment')
1078         self.listModules.insert(tk.END, 'Software Bug')
1079
1080
1081     def addFeedbackTextBox(self):
1082
1083         self.lblFeedback = tk.Label(self, text='Leave your comments in the box below' , justify='left', font=NORMAL_FONT, wraplength=400)
1084         self.lblFeedback.grid(row=3, column=2, columnspan=1, rowspan=1, padx=(0,0), pady=(0,0), sticky="sw")
1085
1086
1087         self.txtbxFeedback = tk.Text (self, width=60, height=6 ,font=SMALL_FONT)
1088         self.scrollFeedback = ttk.Scrollbar(self)
1089
1090         self.txtbxFeedback.configure(yscrollcommand=self.scrollFeedback.set)
1091         self.scrollFeedback.configure( command= self.txtbxFeedback.yview)
1092
1093         self.butClear = ttk.Button(self, text="Clear")
1094         self.butClear.grid(row=3, column=3, columnspan=1, rowspan=1, padx=(0,0), pady=(0,0), sticky="s")
1095         self.butClear.configure(command=lambda: self.txtbxFeedback.delete('0.0', tk.END))
1096
1097         self.txtbxFeedback.grid(row=4, column=2, columnspan=2,  rowspan=2, padx=(0,0), pady=(20,0), sticky="nw")
```

```
1098         self.scrollFeedback.grid(row=4, column=4, columnspan=1,  rowspan=2, padx=(0,0), pady=(20,0), sticky="ns")
1099 
1100         self.emptybox = self.txtbxFeedback.get('0.0', tk.END)
1101 
1102     def checkEntriesAreFilled(self):
1103         completed = True
1104         if self.ckbxAnonymousvar.get() == 0:
1105             if (self.entNumber.get() == '') or (self.entName.get() == ''):
1106                 completed = False
1107 
1108 
1109         selection = self.listModules.curselection()
1110         if (len(selection) == 0):
1111             completed = False
1112 
1113 
1114         charPresent = False
1115         for char in self.txtbxFeedback.get('0.0', tk.END):
1116             if char.isalnum():
1117                 charPresent = True
1118         if completed:
1119             completed = charPresent
1120 
1121         return completed
1122 
1123     def getFeedback(self):
1124         feedbackInfo = []
1125         if self.ckbxAnonymousvar.get() == 0:
1126             feedbackInfo.append(self.entNumber.get())
1127             feedbackInfo.append(self.entName.get())
1128         else:
1129             feedbackInfo.append('Anonymous')
1130             feedbackInfo.append('Anonymous')
1131         selection = self.listModules.curselection()
1132         feedbackInfo.append(self.listModules.get(selection[0]))
1133         text = self.txtbxFeedback.get('0.0', tk.END)
1134         withoutNewLineText = text.replace('\n','|')
1135         feedbackInfo.append(withoutNewLineText)
1136 
1137         return feedbackInfo
1138 
1139     def promptFeedback(self, title, msg):
1140         'Method used to display ask for feedback.'
1141 
1142         popupFeedback = tk.Tk()
1143         popupFeedback.geometry("300x100+600+300")
1144 
1145         popupFeedback.wm_title(title)
1146         labTitle = tk.Label(popupFeedback, text=msg, font=NORMAL_FONT)
1147         labTitle.pack(side="top", pady=10)
1148         butOkay = ttk.Button(popupFeedback, text="Okay", command = popupFeedback.destroy)
1149         butOkay.pack()
1150 
1151     def submitFeedback(self):
1152         if self.checkEntriesAreFilled():
1153             feedbackInfo = self.getFeedback()
1154             try:
1155                 with open('appFeedback.csv', 'a') as csvfile:
1156                     fileWriter = csv.writer(csvfile, delimiter='~')
1157                     fileWriter.writerow(feedbackInfo)
1158                     self.promptFeedback('Submitted', 'Thank you for you feedback!')
1159             except IOError:
1160                 print('error')
1161             else:
1162                 pass
1163         else:
1164             self.promptFeedback("Attention!",'You must complete the feedback \n form before clicking submit .')
1165 
1166     def setCommands(self, controller, menu):
1167         self.butMenu.configure(command=lambda: controller.show_frame(menu))
1168 
1169 #//\\//\\//\\//\\///\\//\\//#
1170 #FEEDBACK REVIEW FRAME CLASS#
1171 #//\\//\\//\\//\\//\\//\\//\#
1172 
1173 class FeedbackReviewFrame(tk.Frame):
1174 
1175     def __init__(self, parent, controller):
1176         tk.Frame.__init__(self, parent)
1177 
1178         self.__column0xpad = 80
1179         self.__row0ypad = 40
1180 
1181         self.__allFeedback = []
1182 
1183         self.lblTitle = tk.Label(self, text='User Feedback', justify='center', font=EXTRA_LARGE_FONT, wraplength=400)
1184         self.lblTitle.grid( row=0, column=0,  columnspan=2,  rowspan=1, padx=(self.__column0xpad,40), pady=(self.__row0ypad,0), sticky="n")
1185 
1186         self.lblMessage = tk.Label(self, text='Here you can look at student feedback about this application.' , justify='left', font=NORMAL_FONT,
    wraplength=400)
1187         self.lblMessage.grid(row=1, column=0, columnspan=2, rowspan=1, padx=(self.__column0xpad,40), pady=(30,0),sticky="sw")
1188 
1189         self.lblName = tk.Label(self, text='Name:' , justify='left', font=NORMAL_FONT, wraplength=300)
1190         self.lblName.grid(row=4, column=0, columnspan=2, rowspan=1, padx=(self.__column0xpad,40), pady=(20,0),sticky="sw")
1191 
1192         self.lblNumber = tk.Label(self, text='Student Number:' , justify='left', font=NORMAL_FONT, wraplength=300)
1193         self.lblNumber.grid(row=5, column=0, columnspan=2, rowspan=1, padx=(self.__column0xpad,40), pady=(20,0),sticky="sw")
1194 
1195         self.lblReference = tk.Label(self, text='Brief Description:' , justify='left', font=NORMAL_FONT, wraplength=300)
1196         self.lblReference.grid(row=6, column=0, columnspan=2, rowspan=1, padx=(self.__column0xpad,40), pady=(20,0),sticky="sw")
1197 
1198         self.addListbox()
1199         self.fillListBoxWithFeedback()
1200 
1201         self.addFeedbackTextBox()
1202 
1203 
1204         self.butMenu = tk.Button(self, text="Back to Menu", font=LARGE_BUTTON_FONT, height= 2, width=10, relief=tk.GROOVE, bg="#C5D0C8")
1205         self.butMenu.bind("<Enter>", lambda event, x=self.butMenu: x.configure(bg="#80A0FF"))
1206         self.butMenu.bind("<Leave>", lambda event, x=self.butMenu: x.configure(bg="#C5D0C8"))
1207         self.butMenu.grid(row=8, column=0, columnspan=5, rowspan=1, padx=(self.__column0xpad,50), pady=(30,0), sticky="s")
```

```
1208
1209     def addFeedbackTextBox(self):
1210
1211         self.lblFeedback = tk.Label(self, text='Feedback:' , justify='left', font=NORMAL_FONT, wraplength=400)
1212         self.lblFeedback.grid(row=2, column=2, columnspan=1, rowspan=1, padx=(30,0), pady=(0,0), sticky="sw")
1213
1214
1215         self.txtbxFeedback = tk.Text (self, width=60, height=10 ,font=SMALL_FONT, state=tk.DISABLED)
1216         self.scrollFeedback = ttk.Scrollbar(self)
1217
1218         self.txtbxFeedback.configure(yscrollcommand=self.scrollFeedback.set)
1219         self.scrollFeedback.configure( command= self.txtbxFeedback.yview)
1220
1221         # self.butClear = ttk.Button(self, text="Clear")
1222         # self.butClear.grid(row=3, column=3, columnspan=1, rowspan=1, padx=(0,0), pady=(0,0), sticky="s")
1223         # self.butClear.configure(command=lambda: self.txtbxFeedback.delete('0.0', tk.END))
1224
1225         self.txtbxFeedback.grid(row=3, column=2, columnspan=2,  rowspan=3, padx=(30,0), pady=(20,0), sticky="nw")
1226         self.scrollFeedback.grid(row=3, column=4, columnspan=1,  rowspan=3, padx=(0,0), pady=(20,0), sticky="ns")
1227
1228         self.emptybox = self.txtbxFeedback.get('0.0', tk.END)
1229
1230     def addListbox(self):
1231
1232         self.listFeedbackLabel = tk.Label(self, text='Select feedback to look at' , justify='left', font=NORMAL_FONT, wraplength=400)
1233         self.listFeedbackLabel.grid(row=2, column=0, columnspan=1, rowspan=1, padx=(self.__column0xpad,0), pady=(20,20), sticky="sw")
1234
1235         self.listFeedback = tk.Listbox(self, height= 3, width=60, font=SMALL_FONT, selectmode=tk.SINGLE)
1236         self.scroll = ttk.Scrollbar(self)
1237
1238         self.listFeedback.configure(yscrollcommand=self.scroll.set)
1239         self.scroll.configure( command= self.listFeedback.yview)
1240
1241         self.listFeedback.selection_set(0, tk.END)
1242         self.listFeedback.focus_set()
1243
1244         self.listFeedback.bind("<<ListboxSelect>>", self.listFeedbackClick)
1245
1246         self.listFeedback.grid(row=3, column=0, columnspan=1, rowspan=1, padx=(self.__column0xpad,0), pady=(20,0), sticky="nw")
1247         self.scroll.grid(row=3, column=1, columnspan=1,  rowspan=1, padx=(0,0), pady=(20,0), sticky="ns")
1248
1249     def fillListBoxWithFeedback(self):
1250         self.listFeedback.delete(0,tk.END)
1251         with open('appFeedback.csv') as csvfile:
1252             rdr = csv.reader(csvfile, delimiter='~')
1253             for row in rdr:
1254                 try:
1255                     self.listFeedback.insert(tk.END, row[2] + ' - ' +row[0])
1256                     self.__allFeedback.append(row)
1257                 except:
1258                     pass
1259
1260     def listFeedbackClick(self, event):
1261
1262         selected = self.listFeedback.curselection()
1263         feedbackIndex = selected[0]
1264         feedback = self.__allFeedback[feedbackIndex]
1265
1266
1267         self.lblName.configure(text = 'Name: ' + feedback[1])
1268         self.lblNumber.configure(text = 'Student Number: ' + feedback[0])
1269         self.lblReference.configure(text = 'Quick Description: ' + feedback[2])
1270         feedbackText = feedback[3]
1271         formattedFeedbackText = feedbackText.replace('|','\n')
1272         self.txtbxFeedback.configure(state=tk.NORMAL)
1273         self.txtbxFeedback.delete('0.0', tk.END)
1274         self.txtbxFeedback.insert('0.0', formattedFeedbackText)
1275         self.txtbxFeedback.configure(state=tk.DISABLED)
1276
1277
1278     def setCommands(self, controller, menu):
1279         self.butMenu.configure(command=lambda: controller.show_frame(menu))
1280
1281
1282 #//\\//\\//\\//\\/#
1283 #EDITOR 001  CLASS#
1284 #//\\//\\//\\//\\/#
1285
1286 class Editor001(EditorFrame):
1287
1288     def __init__(self, parent, controller, mCode):
1289         EditorFrame.__init__(self, parent, controller, mCode)
1290         self.setCommands(controller, HomePage)
1291
1292 #//\\//\\//\\//\\/#
1293 #EDITOR 002  CLASS#
1294 #//\\//\\//\\//\\/#
1295
1296 class Editor002(EditorFrame):
1297
1298     def __init__(self, parent, controller, mCode):
1299         EditorFrame.__init__(self, parent, controller, mCode)
1300         self.setCommands(controller, HomePage)
1301
1302 #//\\//\\//\\//\\//\\//#
1303 #TEST MODULE 001  CLASS#
1304 #//\\//\\//\\//\\//\\//#
1305
1306 class TestModule001 (TestFrame):
1307
1308     def __init__(self, parent, controller, mCode, mName, lCompleted=False):
1309         TestFrame.__init__(self, parent, controller, mCode, mName, lCompleted)
1310         self.setCommands(controller, HomePage)
1311
1312 #//\\//\\//\\//\\//\\//#
1313 #TEST MODULE 002  CLASS#
1314 #//\\//\\//\\//\\//\\//#
1315
1316 class TestModule002 (TestFrame):
1317
1318     def __init__(self, parent, controller, mCode, mName, lCompleted=False):
```

```
        TestFrame.__init__(self, parent, controller, mCode, mName, lCompleted)
        self.setCommands(controller, HomePage)

#//\\//\\//\\//\\//#
#TEST SCORES CLASS#
#//\\//\\//\\//\\//#

class TestScores(TestScoresFrame):

    def __init__(self, parent, controller):
        TestScoresFrame.__init__(self, parent, controller)
        self.setCommands(controller, SearchPage)

#//\\//\\//\\//\\//\#
#SEARCHE PAGE CLASS#
#//\\//\\//\\//\\//\#

class SearchPage(SearchScoresFrame):

    def __init__(self, parent, controller):
        SearchScoresFrame.__init__(self,parent, controller)

#//\\//\\//\\//\\//#
#USER FEEDBACK CLASS#
#//\\//\\//\\//\\//#

class UserFeedback (FeedbackSubmitFrame):

    def __init__(self, parent, controller, moduleList):
        FeedbackSubmitFrame.__init__(self, parent, controller, moduleList)
        self.setCommands(controller, HomePage)

#//\\//\\//\\//\\//\#
#REVIEW FEEDBACK CLASS#
#//\\//\\//\\//\\//\#

class ReviewFeedback(FeedbackReviewFrame):

    def __init__(self, parent, controller):
        FeedbackReviewFrame.__init__(self, parent, controller)
        self.setCommands(controller, HomePage)

#//\\//\\//\\//\\//#
#LOGIN FRAME CLASS#
#//\\//\\//\\//\\/#
class LoginFrame(tk.Frame):
    'Login Frame'

    def __init__(self, parent, controller):
        'Configure the root Window of the app via the self.root attribute.'
        tk.Frame.__init__(self, parent)

        # self.__column0xpad = 150
        # self.__row0ypad = 75

        self.__column0xpad = 40
        self.__row0ypad = 20
        self.lblTitle = tk.Label(self, background='black', text="Welcome to Your Learning Central", justify='center', font=("Helvetica", 30), fg='white',
wraplength=300)
        self.lblTitle.grid( row=0, column=0,  columnspan=2,  rowspan=1, padx=(self.__column0xpad,0), pady=(self.__row0ypad,30), sticky=('N', 'S', 'E', 'W'))

        self.lblMessage = tk.Label(self, text="You are not logged in, Please enter your credentials and click the Login button below.", justify='center',
font=NORMAL_FONT, wraplength=300)
        self.lblMessage.grid(row=1, column=0, columnspan=2, rowspan=1, padx=(self.__column0xpad,0), pady=(0,10), sticky="n")

        self.lblUsername = tk.Label(self, text="Username", justify='left', font=NORMAL_FONT, wraplength=300)
        self.lblUsername.grid(row=2, column=0, columnspan=1, rowspan=1, padx=(self.__column0xpad-30,0), pady=(0,0), sticky="w")

        self.entUsername = ttk.Entry(self, width=35)
        self.entUsername.grid(row=2, column=1, columnspan=1, rowspan=1, padx=(0,0), pady=(0,0), sticky="w")

        self.lblPassword = tk.Label(self, text="Password", justify='left', font=NORMAL_FONT, wraplength=400)
        self.lblPassword.grid(row=3, column=0, columnspan=1, rowspan=1, padx=(self.__column0xpad-30,0), pady=(0,50), sticky="w")

        self.entPassword = ttk.Entry(self, width=35, show='*')
        self.entPassword.grid(row=3, column=1, columnspan=1, rowspan=1, padx=(0,0), pady=(0,50), sticky="w")

        self.butShow = ttk.Button(self, text='Show', command=lambda: self.showPassword())
        self.butShow.grid(row=3, column=2, columnspan=1, rowspan=1, padx=(0,20), pady=(0,50), sticky="w")

        self.butLogin = tk.Button(self, text="Login", font=LARGE_BUTTON_FONT, height= 1, width=10, relief=tk.GROOVE, bg="#C5D0C8")
        self.butLogin.bind("<Enter>", lambda event, x=self.butLogin: x.configure(bg="#80A0FF"))
        self.butLogin.bind("<Leave>", lambda event, x=self.butLogin: x.configure(bg="#C5D0C8"))
        self.butLogin.grid(row=4, column=0, columnspan=2, rowspan=1, padx=(self.__column0xpad+50,0), pady=(0,10), sticky="n")

    def showPassword(self):
        self.entPassword.configure(show='')
        self.butShow.configure(text='Hide', command=lambda: self.hidePassword())

    def hidePassword(self):
        self.entPassword.configure( show='*')
        self.butShow.configure(text='Show', command=lambda: self.showPassword())

#//\\//\\//\\//\\//#
#LOGIN PAGE CLASS#
#//\\//\\//\\//\\//#
class LoginPage (LoginFrame):

    def __init__(self, parent, controller):
        LoginFrame.__init__(self, parent, controller)
        self.butLogin.configure(command=lambda: self.vallogin(controller))

    def vallogin(self, controller):
        username = self.entUsername.get()
        password = self.entPassword.get()
        currentaccount = ""
        users = {}
        checked1 = False
        checked2 = False

        with open("login.csv") as csvfile:
```

```
            rdr = csv.reader(csvfile)
            headers = next(rdr, None)
            for rows in rdr:
                users[rows[3]] = (rows[4], rows[1])
            if username == "" :
                checked1 = True
            if password == "":
                checked2 = True
            if checked2 or checked1:
                tm.showerror("Login error", "Please enter a valid username and password")
            else:
                for k, v in users.items():
                    if username == k and password == v[0]:
                        if v[1] == 'student':
                            controller.geometry("1097x718+100+50")
                            controller.validLogin()
                            controller.show_frame(HomePage)
                            break

                        if v[1] == 'lecturer':
                            controller.geometry("1097x718+100+50")
                            controller.validLogin()
                            controller.show_frame(HomePage)
                            break
                            #
                if (username != k and checked1 == False) or (password != v[0] and checked2 == False):
                    tm.showerror("Login error", "Your username and/or password unvalid")

#//\\//\\//#
#PAGE CLASS#
#//\\//\\//#

EXTRA_LARGE_FONT = ("MS", 22, "bold")
L_FONT = ("Verdana", 12)
LARGE_BUTTON_FONT = ("MS", 14, "bold")
LARGE_FONT= ("MS", 14, "bold")
NORMAL_FONT= ("MS", 12)
SMALL_FONT= ("MS", 10)
POPUP_TITLE_FONT= ("MS", 8, "bold")
POPUP_MESSAGE_FONT= ("MS", 6)

def popupMessage(message):

    popupMsg = tk.Tk()
    popupMsg.geometry("350x150")

    popupMsg.wm_title("Warning!")
    labTitle = tk.Label(popupMsg, text=message)
    labTitle.pack(side="top", pady=10)
    butOkay = ttk.Button(popupMsg, text="Okay", command = popupMsg.destroy)
    butOkay.pack()
    popupMsg.mainloop()

#//\\//\\/\//#
#MODULE CLASS#
#//\//\\//\//#

class Module:


    def __init__(self, code, name, lessonCompleted):
        self.__moduleCode = code
        self.__moduleName = name
        self.__moduleTest = Test(self.__moduleCode)
        self.__lessonCompleted = False

    def lessonNowCompleted(self):
        self.__lessonCompleted = True
        return

    def getModuleCode(self):
        return self.__moduleCode

    def getModuleName(self):
        return self.__moduleName

    def getModuleTest(self):
        return self.__moduleTest

import tkinter
from tkinter import *


class Settings(Frame):

    def __init__(self, parent, controller):
        LoginFrame.__init__(self, parent, controller)
        self.butLogin.configure(command=lambda: self.vallogin(controller))
        self.bgColor = "#EFEFEF"
        self.widgets()

    def widgets(self):
        self.alabel = Label(self, text="SETTINGS", font=("Helvetica", 18, "bold", "italic"))

        self.bgColorLabel = Label(self, text="Background Colour:")

        self.bgColor = StringVar(self)
        self.bgColor.set("Default")
        self.optionBG = OptionMenu(self, self.bgColor, "Default", "White", "Peach")

        self.button = Button(self, text="Apply")
        self.button.config(command = lambda:self.change(self.bgColor.get()))

        self.alabel.pack()
        self.bgColorLabel.pack()
        self.optionBG.pack()
        self.button.pack(pady=(20, 0))

    def change(self, value):
        if value == "Default":
            colourBG = "#EFEFEF"
```

```
1539         elif value == "White":
1540             colourBG = "#FFFFFF"
1541         elif value == "Peach":
1542             colourBG = "#FFDCB9"
1543         self.master.change_background(colourBG)
1544 
1545         # Change page
1546         self.master.change_page('introduction')
1547 
1548 
1549 def main():
1550     # Create window
1551     window = tkinter.Tk()
1552     app = Settings(master=window)
1553     # Window title
1554     window.title("Settings")
1555     # Window size
1556     window.geometry("300x600")
1557     # Window icon
1558     window.wm_iconbitmap('Icon.ico')
1559     # Draw the window, start application
1560     app.mainloop()
1561     window.destroy()
1562 
1563 
1564 #//\\//\\//\\/#
1565 #MAIN FUNCTION#
1566 #//\\//\\//\\/#
1567 
1568 def main():
1569 
1570     app = Application()
1571     app.geometry("417x440+450+90")
1572     app.configure(background = 'blue')
1573     app.mainloop()
1574 
1575 if __name__ == "__main__":
1576     main()
```