Appendix:

https://medium.com/@m.munirahmed113/experience-of-applying-pomodoro-technique-in-my-life-86cd128108e9

https://www.youtube.com/watch?v=5uVXbb1ymVs&list=PLMYBerfrSZNWKyH274K-6PEa1Mmqrj2FL&index=162&t=0s

https://www.youtube.com/playlist?list=PLCC34OHNcOtoC6GglhF3ncJ5rLwQrLGnV

Program code:

```
"""
title: Homework Planner
author: Chelsea Chen
date created: 15-06-20
"""
# imported items
from tkinter import *
import time

#---- Arrays -----#
months = (
("January", 1), ("February", 2), ("March", 3), ("April", 4), ("May", 5), ("June", 6), ("July", 7),
("August", 8),
("September", 9), ("October", 10), ("November", 11), ("December", 12))
monthOptions = ["January",
                "February",
                "March",
                "April",
                "May",
                "June",
                "July",
                "August",
                "September",
                "October",
                "November",
                "December"]

monthDay = {
        "January":31,
        "February":29,
        "March":31,
        "April":30,
        "May":31,
        "June":30,
        "July":31,
        "August":31,
```

```
        "September":30,
        "October":31,
        "November":30,
        "December":31
    }

root = Tk()
root.title("Homework Planner")
root.geometry("500x400")




#--- Processing --- #
# Create Calendar
def cal():
    calendar = []
    for i in range(12): # Create slots for months
        calendar.append([])
        for j in range(31): # Create slots for days in a month
            calendar[i].append([])
            for l in range(24): # Create slots for the hours in a day
                calendar[i][j].append([])
                for m in range(60): # Create slots for minutes in a hour
                    calendar[i][j][l].append([])

    return calendar

date = cal()

dayOptionsFix = []
for i in range(31):
    dayOptionsFix.append(i + 1)
hourOptionsFix = []
for i in range(24):
    hourOptionsFix.append(i)
minuteOptionsFix = []
for i in range(60):
    minuteOptionsFix.append(i)


# save the event
def saveEvent():
```

```
    global editor
    global confirmLabel



    monthStr = monthInput.get() #retrieve month input
    for i in range(len(months)):
        if monthStr == months[i][0]:
            month = int(months[i][1]) - 1 # change data type of month from string to integer

    day = dayInput.get() - 1 #retrive day input
    hour = hourInput.get() #retrieve hour input
    minute = minuteInput.get() #retrieve minute input

    date[month][day][hour][minute] = str(eventInput.get()) #store event in calendar array

    confirmLabel.config(text= "                                                                    ")
    confirmLabel.config(text = "Event " + str(date[month][day][hour][minute]) + " is saved") #print
the that the event is saved
    editor.update()


# view the event
# view the scheduled due date or study time by time
def viewEventByTime():
    global confirmLabel
    global date
    global monthOptions
    global dayOptionsFix
    global hourOptionsFix
    global minuteOptionsFix

    global month
    global day
    global hour
    global minute

    global dayInput
    global hourInput
    global eventInput
    global minuteInput

    monthStr = monthInput.get() # retrieve the month inputted
```

```
    for i in range(len(months)):
        if monthStr == months[i][0]:
            month = int(months[i][1]) - 1 #change the data type of the month and change it from a
string to an integer

    day = int(dayInput.get()) # retrieve the day inputted

    hour = hourInput.get() # retrieve the hour inputted
    minute = minuteInput.get() # retrieve the minute inputted
    eventInput.delete(0, END)
    eventInput.insert(0, date[month][day - 1][hour][minute])

    confirmLabel.config(text="                                                                    ")
    confirmLabel.config(text="Event "+ str(date[month][day - 1][hour][minute]) + " is retrieved")
#output that the event is retrieved
    editor.update()


# view the scheduled due date / study time by name
def viewEventByName():

    global confirmLabel
    global date
    global monthOptions
    global dayOptionsFix
    global hourOptionsFix
    global minuteOptionsFix

    global month
    global day
    global hour
    global minute

    global dayInput
    global hourInput
    global eventInput
    global minuteInput



    confirmLabel.config(text="                                                    ")
    confirmLabel.config(text="Search...")
    editor.update()
```

```
    eventStr = str(eventInput.get()) #retrieve the event name inputed
    findEvent = False # event is not yet found
    month = 0 #make sure no data overlaps
    day = 0
    hour = 0
    minute = 0

    for i in range(12): #run through calendar array
        for j in range(31):
            for l in range(24):
                for m in range(60):
                    print(str(i) + " " + str(j) + " " + str(l) + " " + str(m))
                    if str(date[i][j][l][m]) == eventStr: #if there is an event scheduled with the same name
as inputted
                        month = i
                        day = j
                        hour = l
                        minute = m
                        findEvent= True #event is found
                        break

    print("month" + str(month) + "day" + str(day) + "dayArray:" + str(dayOptionsFix[day]) + "hour"
+ str(
        hour) + "minute" + str(minute))

    confirmLabel.config(text="                                                    ")
    monthInput.set(monthOptions[month]) #set to display the month of the scheduled event
    dayInput.set(dayOptionsFix[day]) #set to display the day of the scheduled event
    hourInput.set(hourOptionsFix[hour]) #set to display the hour of the scheduled event
    minuteInput.set(minuteOptionsFix[minute]) #set to display the minute of the scheduled event
    eventInput.delete(0, END)
    if findEvent:
        eventInput.insert(0, date[month][day][hour][minute])
        confirmLabel.config(text="Event " +str(date[month][day][hour][minute])  + " is retrieved") #
output that the event is retrieved
    else:
        confirmLabel.config(text=eventStr + " does not exist") # if there is no event scheduled
under this name, output that event doesn't exist


def exitEvent():
    editor.destroy() #exit the event window
```

```
def deleteEvent():
    global confirmLabel
    monthStr = monthInput.get() # retrieve the month inputted
    for i in range(len(months)):
        if monthStr == months[i][0]:
            month = int(months[i][1]) - 1 #change the data type from a string to an integer and
subtract one for placement

    day = dayInput.get() - 1 # get day inputted and subtract one for placement in the calendar
array
    hour = hourInput.get() # get the hour inputted
    minute = minuteInput.get() # get the minute inputted

    date[month][day][hour][minute] = "" #delete item in array
    eventInput.delete(0, END)
    month = 0
    day = 0
    hour = 0
    minute = 0
    monthInput.set(monthOptions[month]) # set month to zero
    dayInput.set(dayOptionsFix[day]) # set day to zero
    hourInput.set(hourOptionsFix[hour]) # set hour to zero
    minuteInput.set(minuteOptionsFix[minute]) # set minute to zero

    confirmLabel.config(text="                                                    ")
    confirmLabel.config(text="event deleted") #output that the event was deleted

# exit out of the timer
def exitTimer():

    global exitStudy

    exitStudy = True
    editor2.destroy()


# initiated when start timer button pressed
def startTimer():
    global editor2
    global timerLabel
    global studytime
```

```
    global study
    global exitStudy

    exitStudy = False


    while not exitStudy:
        for i in range(1, 5): # repeat four times
            if exitStudy:
                break
            studyperiod = 1500 #set to 25 minutes (1500 seconds in 25 minute)

            #25 minutes study
            while studyperiod > 0:
                if exitStudy:
                    break
                minutes = studyperiod // 60 # get minutes for timer
                seconds = studyperiod % 60 # get seconds for timer
                print(studyperiod)
                print(seconds)
                studytime.config(text = "                                                                    ")
                studytime.config(text = "Start studying for 25 minutes!") #ouput to start studying

                timerLabel.config(text="                                    ")
                timerLabel.config(text = str(minutes).zfill(2) + ":" + str(seconds).zfill(2)) #output the
timer
                editor2.update()
                time.sleep(1) # each second the time goes down by a second
                studyperiod = studyperiod - 1



            #5 minutes short break
            shortbreak = 300 # set to five minutes (300s in a minute)
            if not exitStudy:
                studytime.config(text = "                                                                    ")
                studytime.config(text = "Start your 5 minute break! Take this time to destress")
#output to start studying

            while shortbreak >0:
                if exitStudy:
                    break
                minutes = shortbreak // 60 # get minutes for timer
```

```
            seconds = shortbreak % 60 # get seconds for timer

            timerLabel.config(text="                                    ")
            timerLabel.config(text=str(minutes).zfill(2) + ":" + str(seconds).zfill(2)) #Output timer
(minutes and seconds)
            editor2.update()
            time.sleep(1) # each second the time goes down by a second
            shortbreak = shortbreak - 1


        #20 minutes long break
        if not exitStudy:
            studytime.config(text="                                                    ")
            studytime.config(text="Start your 20 minute break!")

        longbreak = 5
        while longbreak > 0:
            if exitStudy:
                break
            minutes = longbreak // 60
            seconds = longbreak % 60

            timerLabel.config(text="                                    ")
            timerLabel.config(text=str(minutes).zfill(2) + ":" + str(seconds).zfill(2))
            editor2.update()
            time.sleep(1)
            longbreak = longbreak - 1




def pomTimer():
    global editor2
    global studytime
    global timerLabel

    editor2 = Tk()
    editor2.title("Pomodoro Timer") # title the window
    editor2.geometry("550x300")
    startTimerButton = Button(editor2, text="Press to start", command=startTimer) #create button
for starting the timer
    startTimerButton.grid(row=1,column=0,columnspan = 2,padx=180,pady=(20,20), sticky=W)
    studytime = Label(editor2, text="            ")
```

```
    studytime.grid(row=2, column=1, sticky="w")
    timerLabel = Label(editor2, text="              ")
    timerLabel.grid(row=3, column=0, columnspan=2, padx=180, sticky=W)

    exitTimerButton = Button(editor2, text = "Exit Timer", command = exitTimer) #create button
for exiting the timer
    exitTimerButton.grid(row=6, column=0, columnspan =2,padx=180,sticky=W)




def planEvent(choice):
    global editor

    global confirmLabel

    global monthInput
    global dayInput
    global hourInput
    global eventInput
    global minuteInput

    global day
    global monthDropdown
    global dayDropDown
    global monthDay

    day = 0

    def monthChanged(*args):

        global dayInput
        global dayDropDown
        global editor
        global day
        global monthDay
        global monthInput
# get the length of each month
        dayInput = IntVar()
        dayOptions = []
        for i in range(monthDay[monthInput.get()]):
            dayOptions.append(i + 1)
```

```
        dayInput.set(dayOptionsFix[day])
        dayDropdown = OptionMenu(editor, dayInput, *dayOptions)
        dayDropdown.grid(row=2, column=3, sticky="e")

    editor = Toplevel(root)
# name the window event
    editor.title("Event")
    editor.geometry("550x300")

    confirmLabel = Label(editor, text="")
    eventLabel = Label(editor, text="Name of the Event:") #create label for textbox
    eventLabel.grid(row=0, column=0, columnspan=8, sticky="w")
    eventInput = Entry(editor, width=50) # create textbox
    eventInput.grid(row=1, column=0, columnspan=8, sticky="w")

    monthLabel = Label(editor, text="Month:") # create label for the month
    monthLabel.grid(row=2, column=0, sticky="w")

    monthInput = StringVar()
    monthInput.trace("w",monthChanged)
    monthDropdown = OptionMenu(editor, monthInput, *monthOptions) #create dropdown menu
for the month
    monthDropdown.grid(row=2, column=1, sticky="e")

    dayLabel = Label(editor, text="Day:") # create label for the day
    dayLabel.grid(row=2, column=2, sticky="w")

    monthInput.set(monthOptions[0])

    hourLabel = Label(editor, text="Hour:") # create label for the hour
    hourLabel.grid(row=2, column=4, sticky=W)
# create hour dropdown menu
    hourOptions = []
    for i in range(24):
        hourOptions.append(i)
    hourInput = IntVar()
    hourInput.set(hourOptions[0])
    hourDropdown = OptionMenu(editor, hourInput, *hourOptions) #make options based on array
    hourDropdown.grid(row=2, column=5, sticky=W)

    minuteLabel = Label(editor, text="Minute:") # create label for the minute
    minuteLabel.grid(row=2, column=6, sticky=W)
# create minute dropdown menu
```

```
    minuteOptions = []
    for i in range(60):
        minuteOptions.append(i)
    minuteInput = IntVar()
    minuteInput.set(minuteOptions[0])
    minuteDropdown = OptionMenu(editor, minuteInput, *minuteOptions) #make options based
on array
    minuteDropdown.grid(row=2, column=7, sticky=W)
    confirmLabel = Label(editor, text="                                                    ", bd=1,
relief=SUNKEN,anchor=W)
    confirmLabel.grid(row=5, column=0, columnspan=8, sticky=W + E)
    if choice == 1: #if schedule due date/ study time window
        saveButton = Button(editor, text="Save Event", command=saveEvent)  #save event button
        saveButton.grid(row=3, column=0, columnspan=2, pady=10)
        confirmLabel.config( text="Please enter the event you want to plan") #instructions for user
input


    if choice == 2 or choice == 3:  #if viewing or deleting scheduled due date/ study time window
        viewButtonByTime = Button(editor, width=15, text="View By Time",
command=viewEventByTime) #view by time button
        viewButtonByTime.grid(row=3, column=0, columnspan=2, pady=10)
        viewButtonByEvent = Button(editor, width=15, text="View By Name",
command=viewEventByName) #view by name button
        viewButtonByEvent.grid(row=3, column=2, columnspan=2, pady=10)
        confirmLabel.config(text="Please enter the event you want to view") # instructions for the
user input

    if choice == 3: #if deleting scheduled due date/ study time window
        deleteButton = Button(editor, text="Delete Event", command=deleteEvent) #delete event
button
        deleteButton.grid(row=3, column=4, columnspan=2, pady=10)
        confirmLabel.config(text="Please enter the event you want to delete") #instructions for user
input



    exitButton = Button(editor, text="Exit", command=exitEvent) #button to exit
    exitButton.grid(row=3, column=6, columnspan=2, pady=10)
    editor.mainloop()

def exitProgram():
    root.destroy()
```

```
#---- Main ---- #
# Menu
# Main title
welcomeLabel = Label(root, text="Welcome to the Homework Planner, Press one option to
start:") #label for title
welcomeLabel.grid(row=0, column=0, pady=20, columnspan=2)

# choose schedule a due date or studytime
choiceOne = Button(root, text="Schedule due date/study time  ", command=lambda:
planEvent(1)) # button for scheduling a due date/study time
choiceOne.grid(row=1, column=0, padx=90, sticky=W)

# choose to view a due date or studytime
choiceTwo = Button(root, text="View a Scheduled due date/study time  ", command=lambda:
planEvent(2)) # button for viewing scheduled event
choiceTwo.grid(row=2, column=0, padx=90, sticky=W)

# choose to delete a saved due date or studytime
choiceThree = Button(root, text="Delete a scheduled due date/study time", command=lambda:
planEvent(3)) #button for deleting scheduled event
choiceThree.grid(row=3, column=0, padx=90, sticky=W)

# choose to use the pomodoro timer
choiceFour = Button(root, text= "Use the Pomodoro Timer", command=pomTimer) # button for
pomodoro timer
choiceFour.grid(row=4, column =0, padx=90, sticky=W)

# choose to exit the application
choiceFive = Button(root, text= "Exit Application", command = exitProgram) #button for exiting
the application
choiceFive.grid(row=5, column =0, padx=90, sticky=W)
root.mainloop()
```