Code Review of The Software Project:
Productivity Software: Study Outline

Course Title: Software Development Project
Course ID: CSE- 3106

Project By:

M.d. Ashiquzzman Rahad
Student ID: 210201
Jannatul Ferdous Nijhum
Student ID: 210239

Reviewed By:

Sk Miraz Rahman Ani
Student ID: 210211
Mohommad Ibnsina
Student ID: 200210

Submitted To:

Dr. Amit Kumar Mondal
Associate Professor
Computer Science & Engineering Discipline
Khulna University,
Khulna.

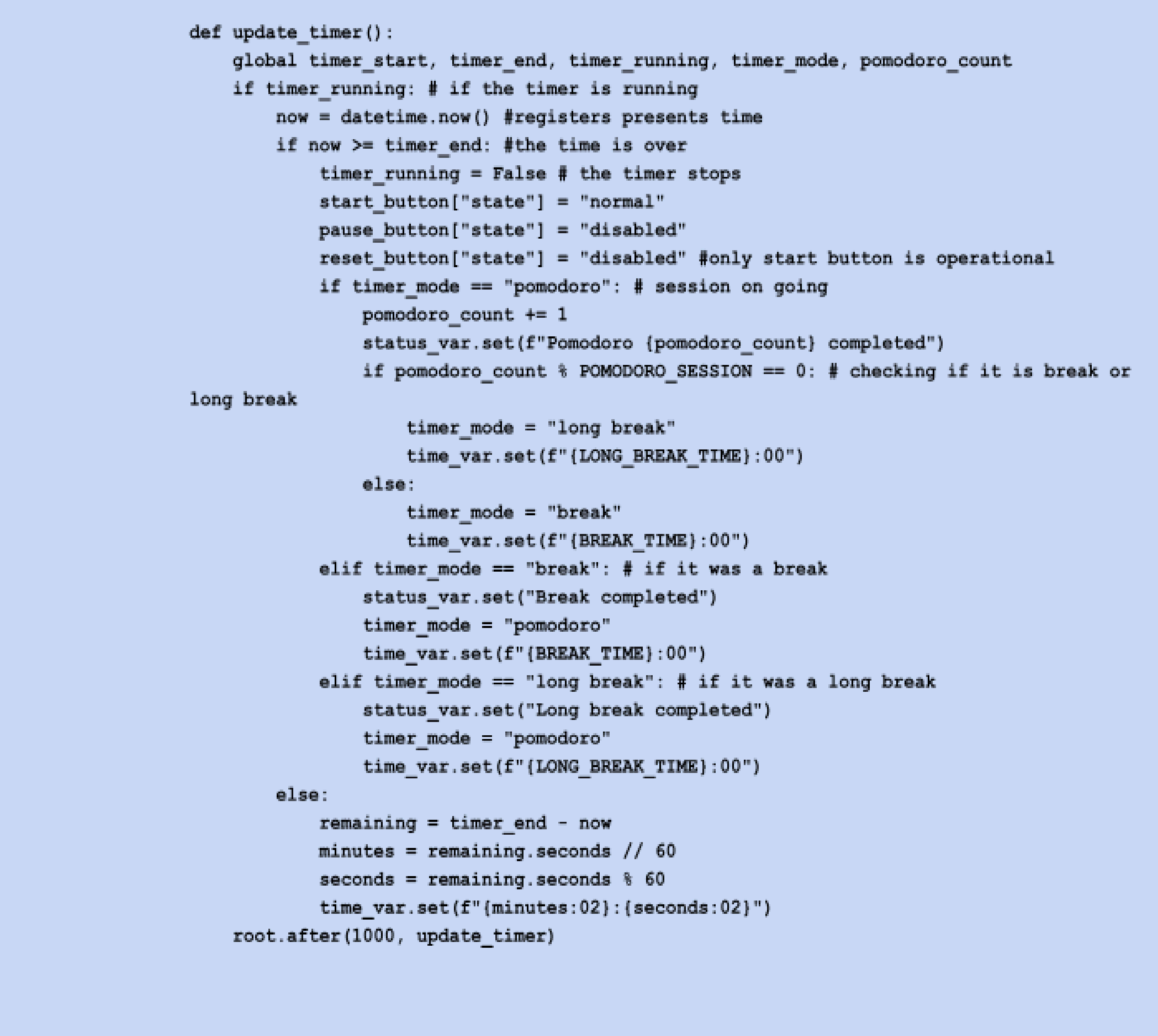
Introduction

This code review evaluates the productivity application: **Study Outline**. The review identifies areas for improvement, adherence to best practices, and suggestions for enhancing maintainability, security, and overall code quality. In this code review, bad smells of the code, architecture evaluation, modularity checking, condition statements of the code & other related sections are evaluated.

Code Smells

1. **Large or complex methods:**

   In terms of method size and complexity, the codebase generally maintains a balance, with few instances of excessively large or intricate methods that might pose readability challenges. On average, methods consist of around 10 lines of code, adhering closely to the standard size. The largest method, found in the pomodoro.py module, spans approximately 30 lines, notably in the update_timer function. Conversely, the syllabus.py module predominantly features smaller methods, contributing to a more streamlined and modular structure overall.



```
def update_timer():
    global timer_start, timer_end, timer_running, timer_mode, pomodoro_count
    if timer_running: # if the timer is running
        now = datetime.now() #registers presents time
        if now >= timer_end: #the time is over
            timer_running = False # the timer stops
            start_button["state"] = "normal"
            pause_button["state"] = "disabled"
            reset_button["state"] = "disabled" #only start button is operational
            if timer_mode == "pomodoro": # session on going
                pomodoro_count += 1
                status_var.set(f"Pomodoro {pomodoro_count} completed")
                if pomodoro_count % POMODORO_SESSION == 0: # checking if it is break or
long break
                    timer_mode = "long break"
                    time_var.set(f"{LONG_BREAK_TIME}:00")
                else:
                    timer_mode = "break"
                    time_var.set(f"{BREAK_TIME}:00")
            elif timer_mode == "break": # if it was a break
                status_var.set("Break completed")
                timer_mode = "pomodoro"
                time_var.set(f"{BREAK_TIME}:00")
            elif timer_mode == "long break": # if it was a long break
                status_var.set("Long break completed")
                timer_mode = "pomodoro"
                time_var.set(f"{LONG_BREAK_TIME}:00")
        else:
            remaining = timer_end - now
            minutes = remaining.seconds // 60
            seconds = remaining.seconds % 60
            time_var.set(f"{minutes:02}:{seconds:02}")
    root.after(1000, update_timer)
```

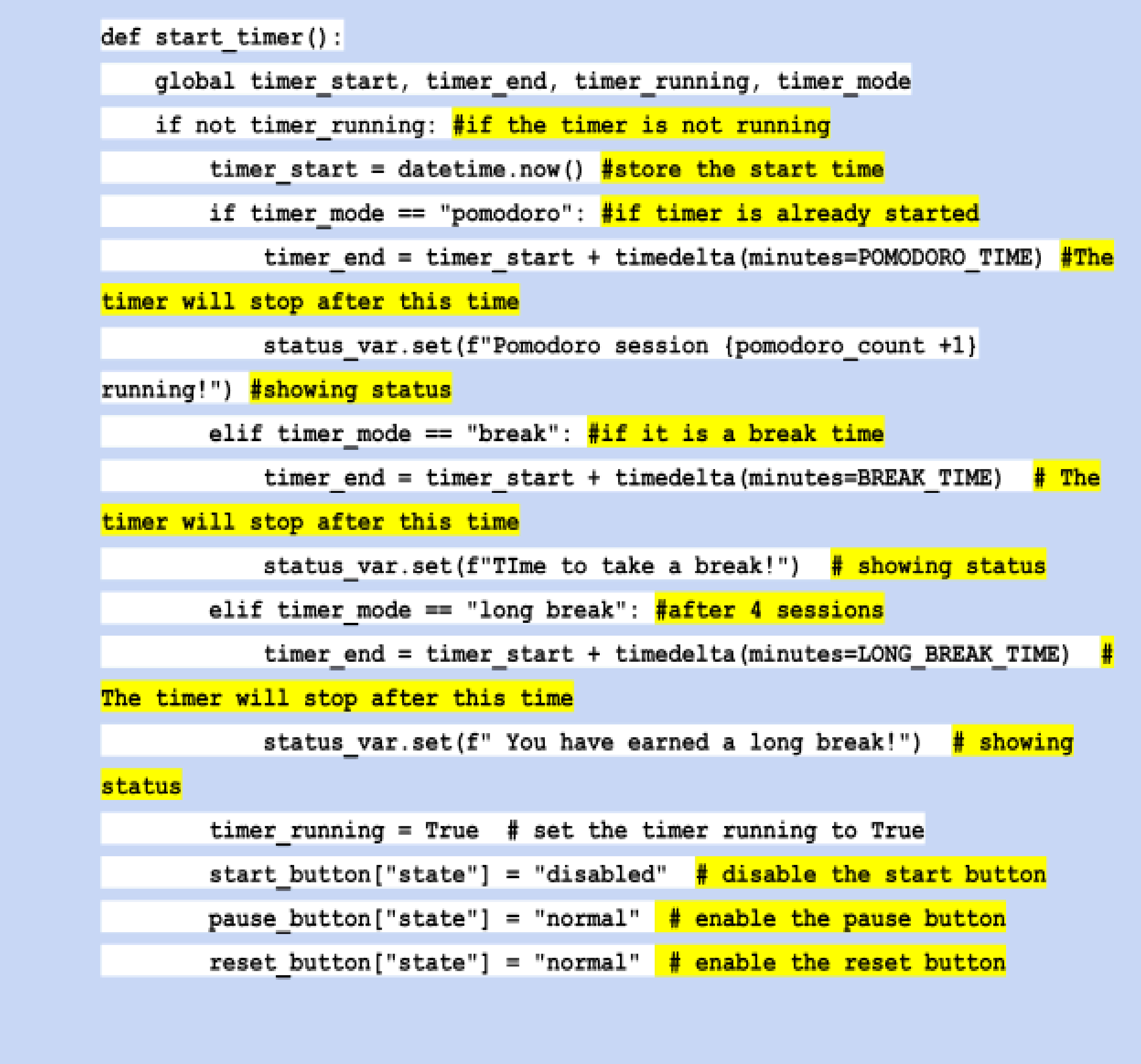
2. Long parameter lists:

There is no method with long parameter lists.

3. Excessive comments:

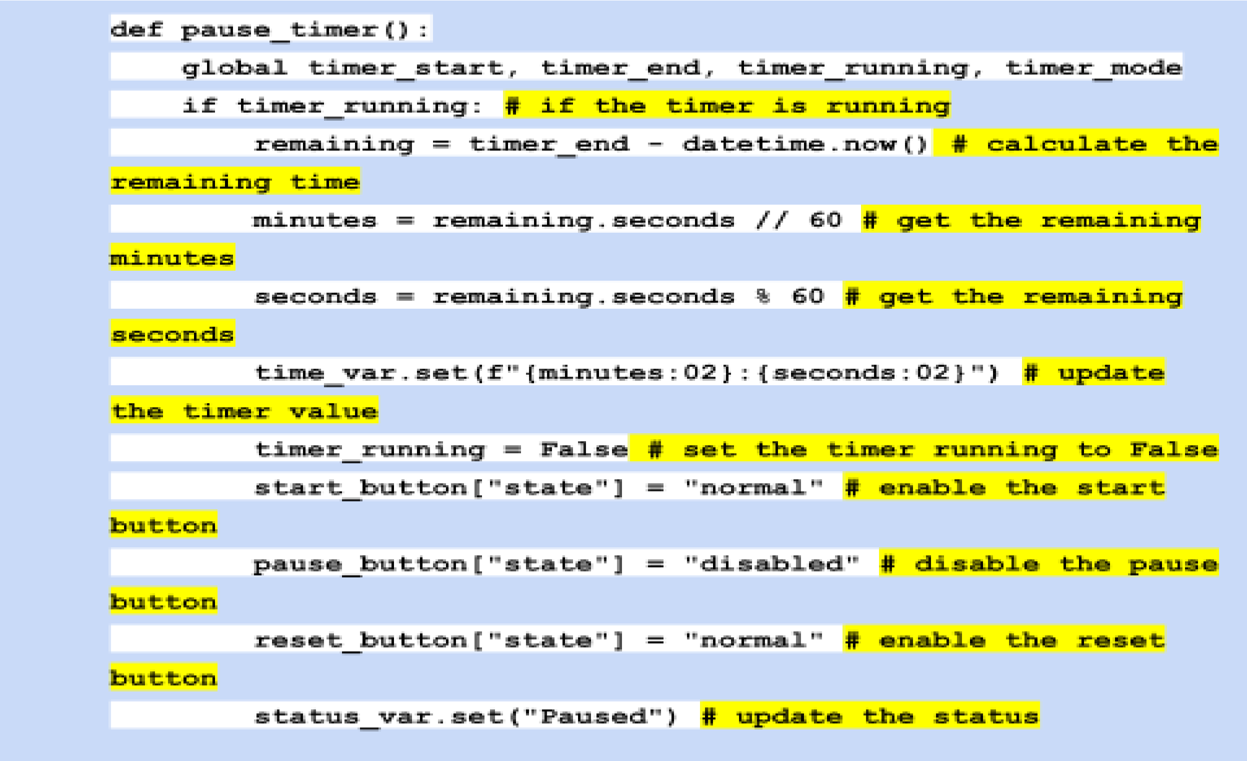
Inconsistencies in the usage of comments are apparent across various modules. While "goal.py" and "pomodoro.py" are inundated with excessive comments, "syllabus.py" stands in stark contrast with not a single comment to be found.

In the start_timer function-



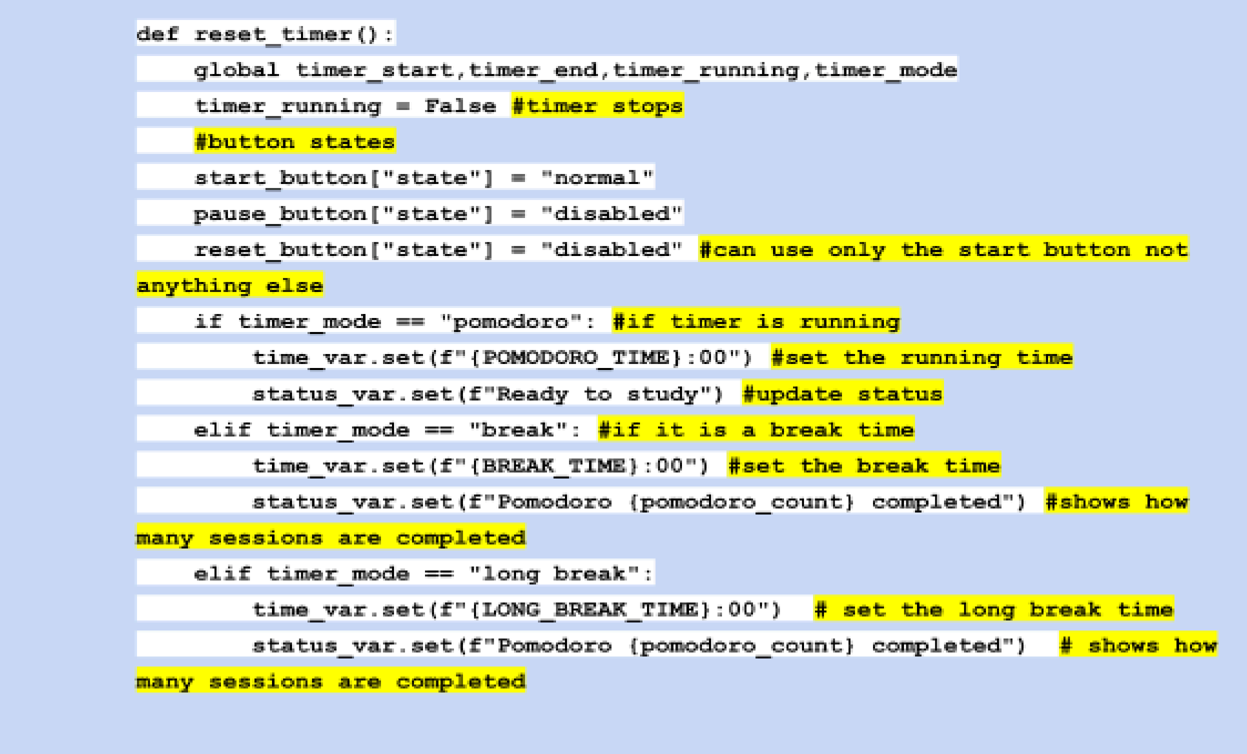
```
def start_timer():
    global timer_start, timer_end, timer_running, timer_mode
    if not timer_running: #if the timer is not running
        timer_start = datetime.now() #store the start time
        if timer_mode == "pomodoro": #if timer is already started
            timer_end = timer_start + timedelta(minutes=POMODORO_TIME) #The
timer will stop after this time
            status_var.set(f"Pomodoro session {pomodoro_count +1}
running!") #showing status
        elif timer_mode == "break": #if it is a break time
            timer_end = timer_start + timedelta(minutes=BREAK_TIME)  # The
timer will stop after this time
            status_var.set(f"TIme to take a break!")  # showing status
        elif timer_mode == "long break": #after 4 sessions
            timer_end = timer_start + timedelta(minutes=LONG_BREAK_TIME)  #
The timer will stop after this time
            status_var.set(f" You have earned a long break!")  # showing
status
        timer_running = True  # set the timer running to True
        start_button["state"] = "disabled"  # disable the start button
        pause_button["state"] = "normal"  # enable the pause button
        reset_button["state"] = "normal"  # enable the reset button
```

In the pause_timer function-



```
def pause_timer():
    global timer_start, timer_end, timer_running, timer_mode
    if timer_running: # if the timer is running
        remaining = timer_end - datetime.now() # calculate the remaining time
        minutes = remaining.seconds // 60 # get the remaining minutes
        seconds = remaining.seconds % 60 # get the remaining seconds
        time_var.set(f"{minutes:02}:{seconds:02}") # update the timer value
        timer_running = False # set the timer running to False
        start_button["state"] = "normal" # enable the start button
        pause_button["state"] = "disabled" # disable the pause button
        reset_button["state"] = "normal" # enable the reset button
        status_var.set("Paused") # update the status
```

In the reset_timer function-



```
def reset_timer():
    global timer_start,timer_end,timer_running,timer_mode
    timer_running = False #timer stops
    #button states
    start_button["state"] = "normal"
    pause_button["state"] = "disabled"
    reset_button["state"] = "disabled" #can use only the start button not anything else
    if timer_mode == "pomodoro": #if timer is running
        time_var.set(f"{POMODORO_TIME}:00") #set the running time
        status_var.set(f"Ready to study") #update status
    elif timer_mode == "break": #if it is a break time
        time_var.set(f"{BREAK_TIME}:00") #set the break time
        status_var.set(f"Pomodoro {pomodoro_count} completed") #shows how many sessions are completed
    elif timer_mode == "long break":
        time_var.set(f"{LONG_BREAK_TIME}:00")  # set the long break time
        status_var.set(f"Pomodoro {pomodoro_count} completed")  # shows how many sessions are completed
```

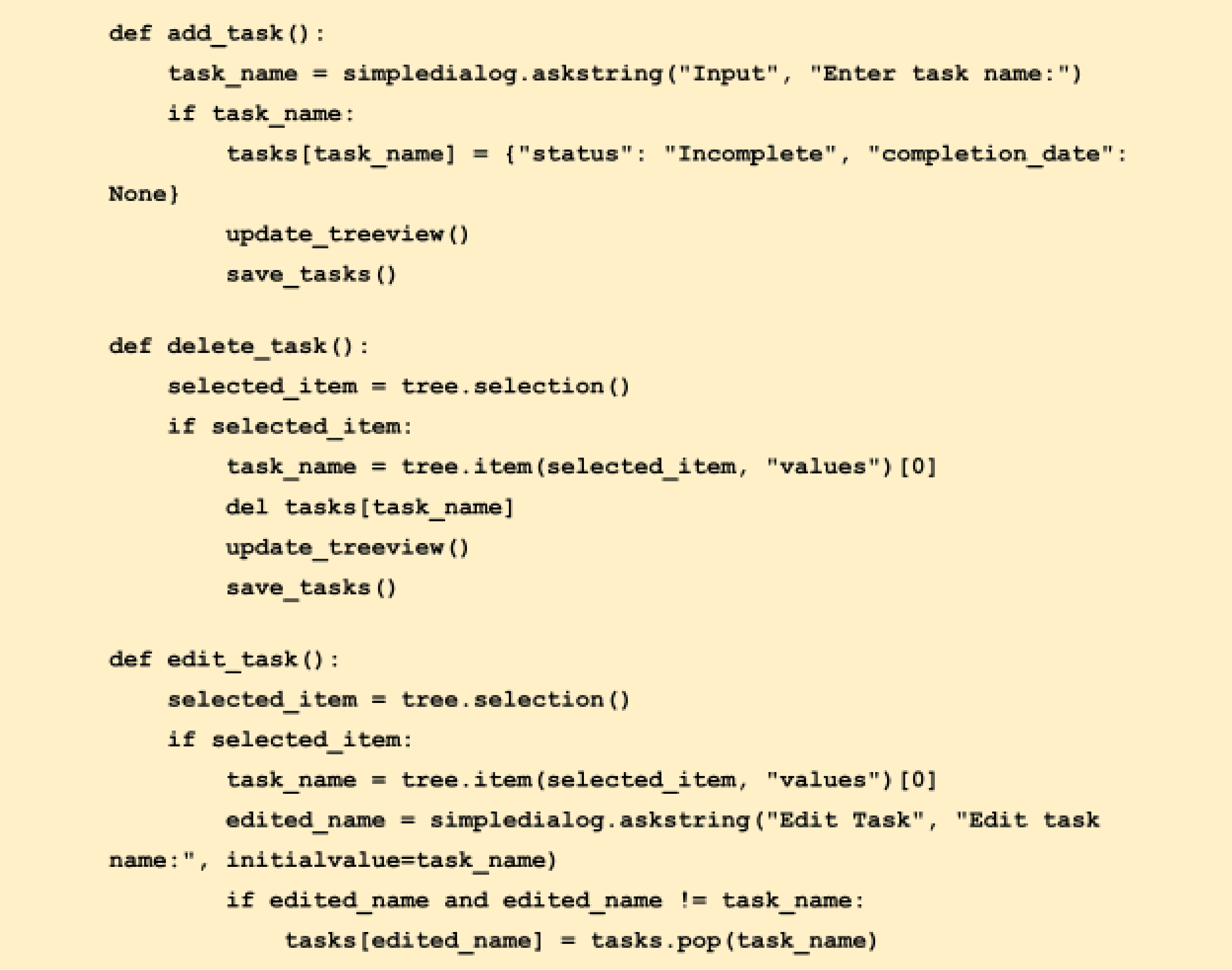
There are too many comments in the methods which are unnecessary.

4. **Duplicate code:**

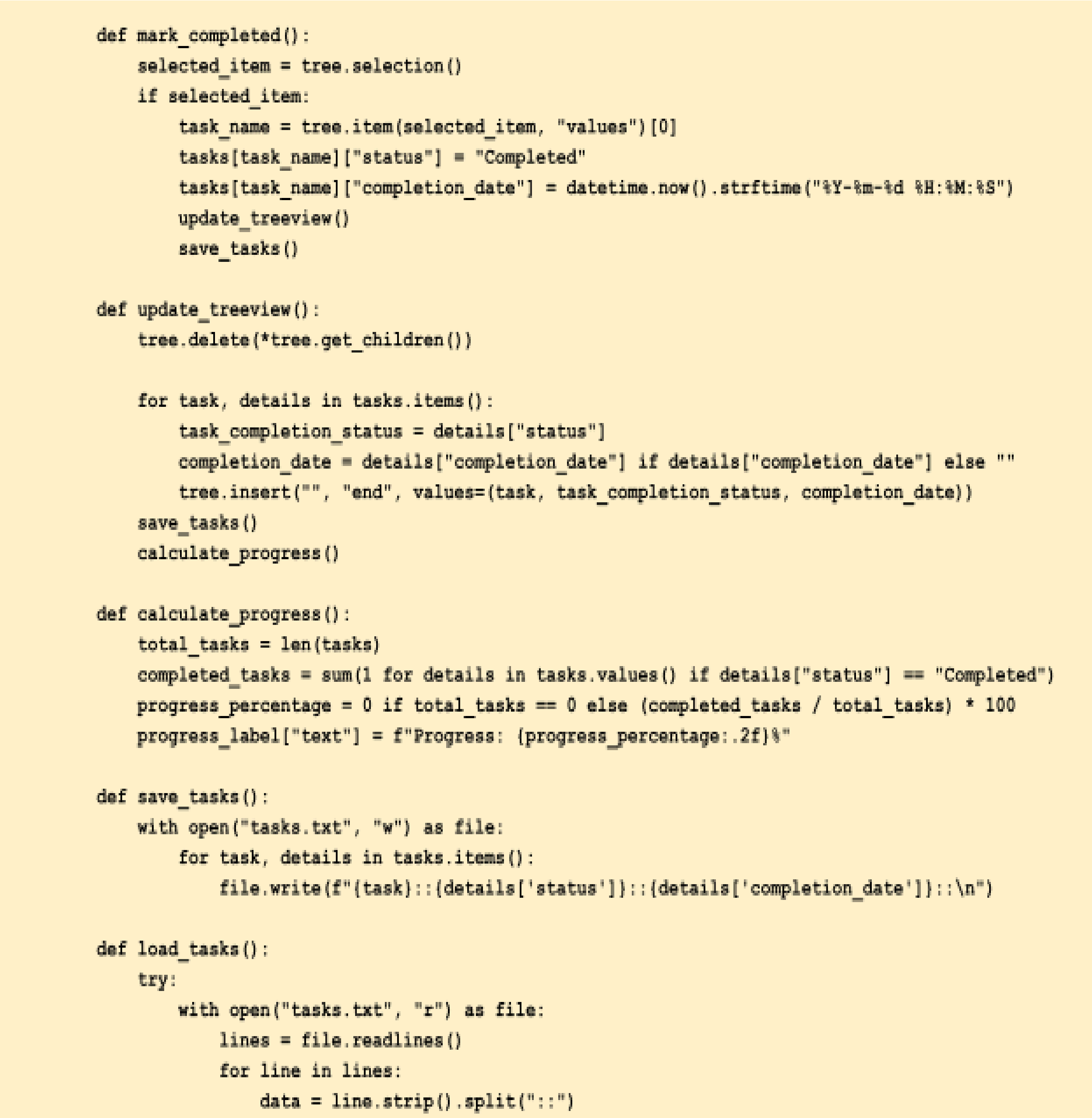
There is no duplicate code in the methods. Effective code reusability has been achieved across the majority of modules.

5. **Inconsistent naming conventions:**

Naming conventions, in many instances, adhere to standardized practices. However, there are instances where variations exist, leading to diverse approaches in naming. For example: update_timer(), save_tasks(), etc.



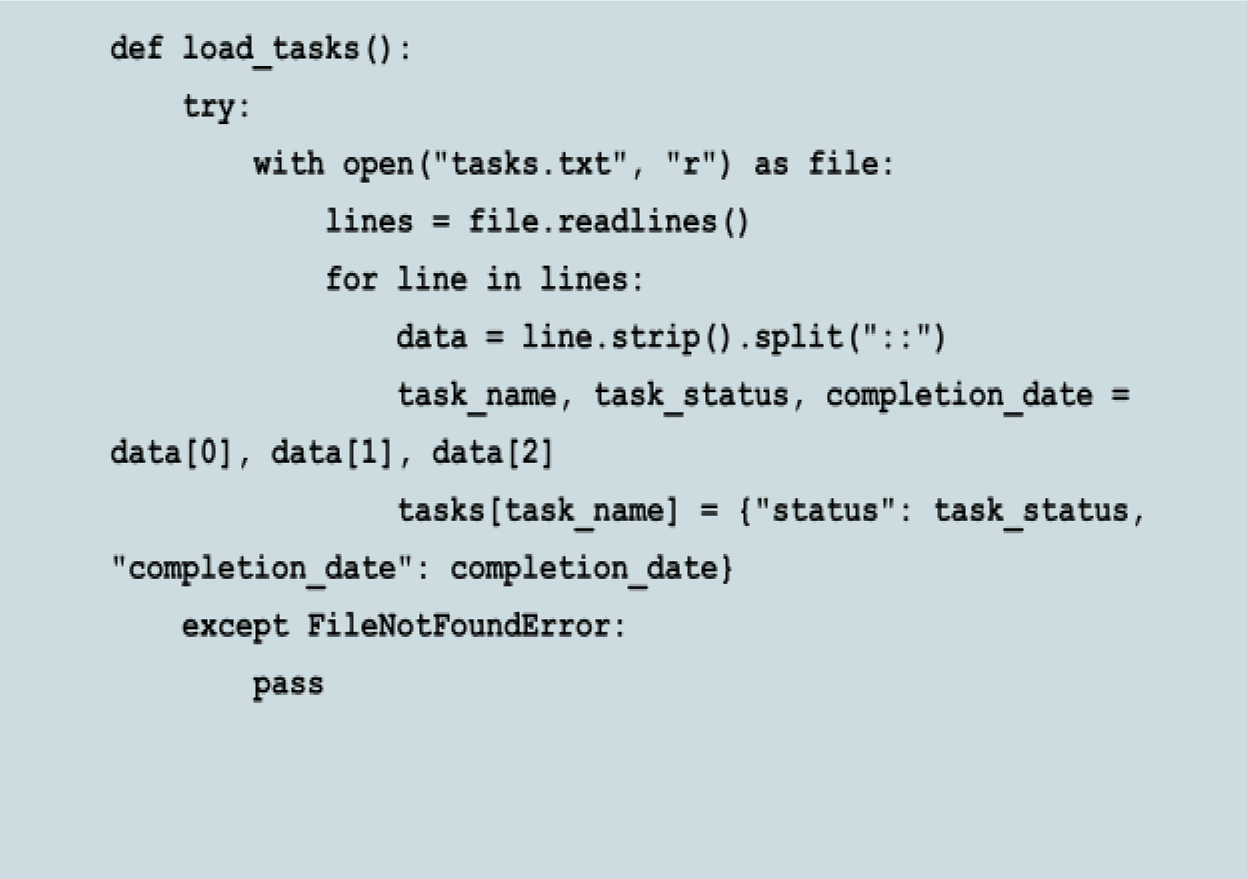
```
def add_task():
    task_name = simpledialog.askstring("Input", "Enter task name:")
    if task_name:
        tasks[task_name] = {"status": "Incomplete", "completion_date":
None}
        update_treeview()
        save_tasks()

def delete_task():
    selected_item = tree.selection()
    if selected_item:
        task_name = tree.item(selected_item, "values")[0]
        del tasks[task_name]
        update_treeview()
        save_tasks()

def edit_task():
    selected_item = tree.selection()
    if selected_item:
        task_name = tree.item(selected_item, "values")[0]
        edited_name = simpledialog.askstring("Edit Task", "Edit task
name:", initialvalue=task_name)
        if edited_name and edited_name != task_name:
            tasks[edited_name] = tasks.pop(task_name)
```



```
def mark_completed():
    selected_item = tree.selection()
    if selected_item:
        task_name = tree.item(selected_item, "values")[0]
        tasks[task_name]["status"] = "Completed"
        tasks[task_name]["completion_date"] = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        update_treeview()
        save_tasks()

def update_treeview():
    tree.delete(*tree.get_children())

    for task, details in tasks.items():
        task_completion_status = details["status"]
        completion_date = details["completion_date"] if details["completion_date"] else ""
        tree.insert("", "end", values=(task, task_completion_status, completion_date))
    save_tasks()
    calculate_progress()

def calculate_progress():
    total_tasks = len(tasks)
    completed_tasks = sum(1 for details in tasks.values() if details["status"] == "Completed")
    progress_percentage = 0 if total_tasks == 0 else (completed_tasks / total_tasks) * 100
    progress_label["text"] = f"Progress: {progress_percentage:.2f}%"

def save_tasks():
    with open("tasks.txt", "w") as file:
        for task, details in tasks.items():
            file.write(f"{task}::{details['status']}::{details['completion_date']}::\n")

def load_tasks():
    try:
        with open("tasks.txt", "r") as file:
            lines = file.readlines()
            for line in lines:
                data = line.strip().split("::")
```

6. Incomplete error handling:

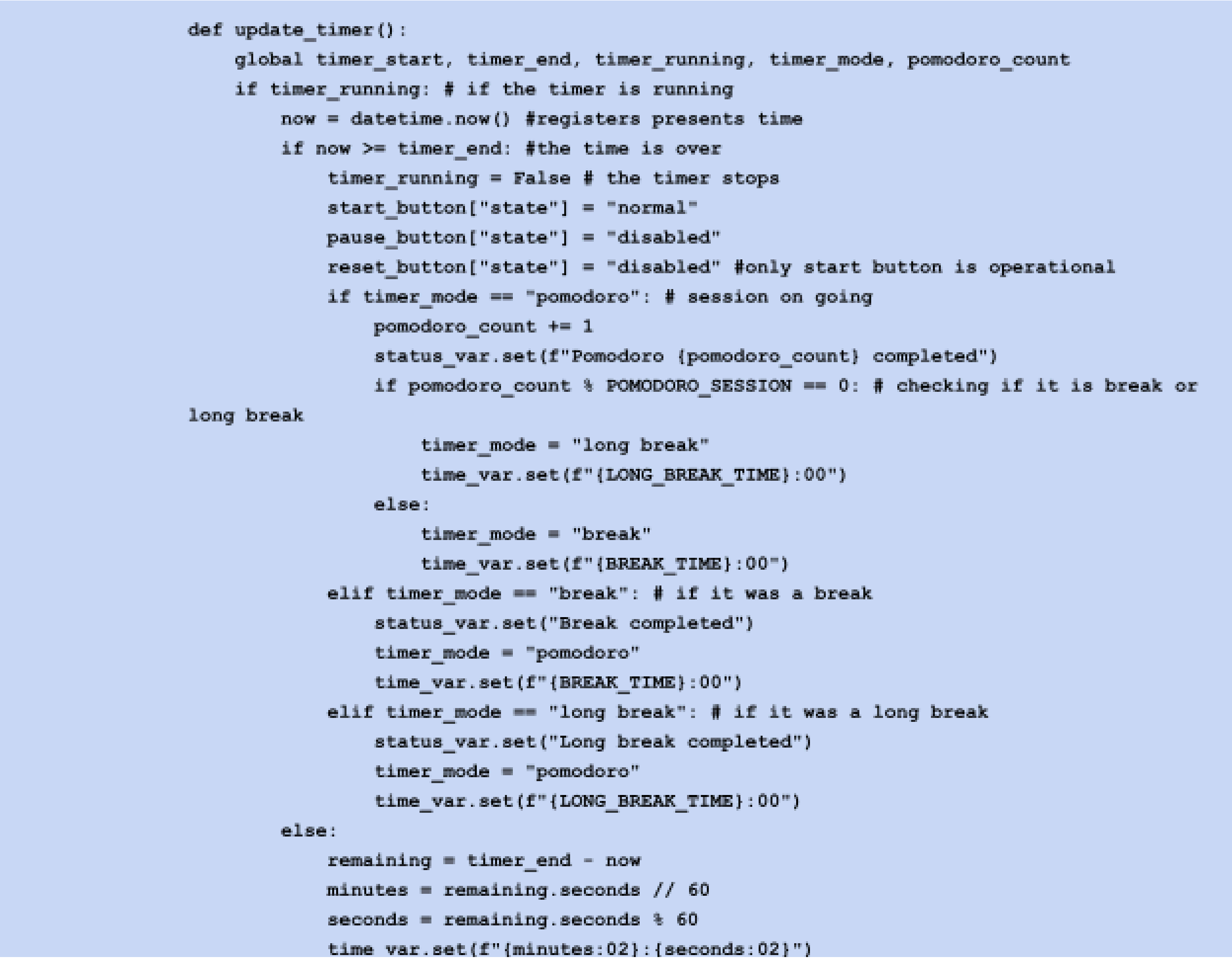
Errors are primarily managed within each module, with special attention paid to file operations to mitigate potential issues. These operations appear to be handled meticulously, minimizing the likelihood of encountering errors.



```
def load_tasks():
    try:
        with open("tasks.txt", "r") as file:
            lines = file.readlines()
            for line in lines:
                data = line.strip().split("::")
                task_name, task_status, completion_date =
data[0], data[1], data[2]
                tasks[task_name] = {"status": task_status,
"completion_date": completion_date}
    except FileNotFoundError:
        pass
```

7. **Too many if/else statements:**

   In the pomodoro.py module, there are some excessive use of if/else statements. But in other modules, the usage of if/else statements are moderated.



```
def update_timer():
    global timer_start, timer_end, timer_running, timer_mode, pomodoro_count
    if timer_running: # if the timer is running
        now = datetime.now() #registers presents time
        if now >= timer_end: #the time is over
            timer_running = False # the timer stops
            start_button["state"] = "normal"
            pause_button["state"] = "disabled"
            reset_button["state"] = "disabled" #only start button is operational
            if timer_mode == "pomodoro": # session on going
                pomodoro_count += 1
                status_var.set(f"Pomodoro {pomodoro_count} completed")
                if pomodoro_count % POMODORO_SESSION == 0: # checking if it is break or
long break
                    timer_mode = "long break"
                    time_var.set(f"{LONG_BREAK_TIME}:00")
                else:
                    timer_mode = "break"
                    time_var.set(f"{BREAK_TIME}:00")
            elif timer_mode == "break": # if it was a break
                status_var.set("Break completed")
                timer_mode = "pomodoro"
                time_var.set(f"{BREAK_TIME}:00")
            elif timer_mode == "long break": # if it was a long break
                status_var.set("Long break completed")
                timer_mode = "pomodoro"
                time_var.set(f"{LONG_BREAK_TIME}:00")
        else:
            remaining = timer_end - now
            minutes = remaining.seconds // 60
            seconds = remaining.seconds % 60
            time_var.set(f"{minutes:02}:{seconds:02}")
```

8. Poor use of inheritance:

In this project, there isn't a singular instance of inheritance misuse that's causing significant issues.

9. Unnecessary dependencies:

There are various unnecessary external libraries and frameworks imported in various modules in this project which are not being used. For example, in pomodoro.py module -

```
import tkinter as tk
from tkinter import *
import ttkbootstrap as tb
from ttkbootstrap.constants import *
from ttkbootstrap import Style
from datetime import datetime, timedelta
import json
from ttkbootstrap.dialogs import Messagebox
import main
from main import frame1, root
```

In the goal.py module-

```
import tkinter as tk
from tkinter import *
import ttkbootstrap as tb
from ttkbootstrap.constants import *
from ttkbootstrap import Style
from datetime import datetime, timedelta
import json

from ttkbootstrap.dialogs import Messagebox
import main
from main import frame1, root
```

These libraries/ frameworks are not being used but still imported unnecessary

10. Magic numbers or hard-coded values:

In most of the cases, there is no sign of hard coding or magic numbers. Instead, contents are introduced and used in the project.

Proposed Architecture Evaluation

The proposed architecture of the project is **“Layered Architecture”**. In the project, we can see the reflection of the proposed architecture. There are different layers or modules in the project.

There is a **main.py** module which has the main window with different frames for the application, which serves as the basic user interface layer. The detailed user interface is actually enclosed in each of the modules.

The **pomodoro.py, goal.py and the syllabus.py** modules serves as the main application functionality layer. These modules have their individual application functionalities included in them. The **study.py** module serves the application functionality of launching the application as a whole.

In each of the module, there are required file operations included which serve as the locally data storing layer.

So, we can say that the project reflects the proposed **Layered Architecture** more or less.

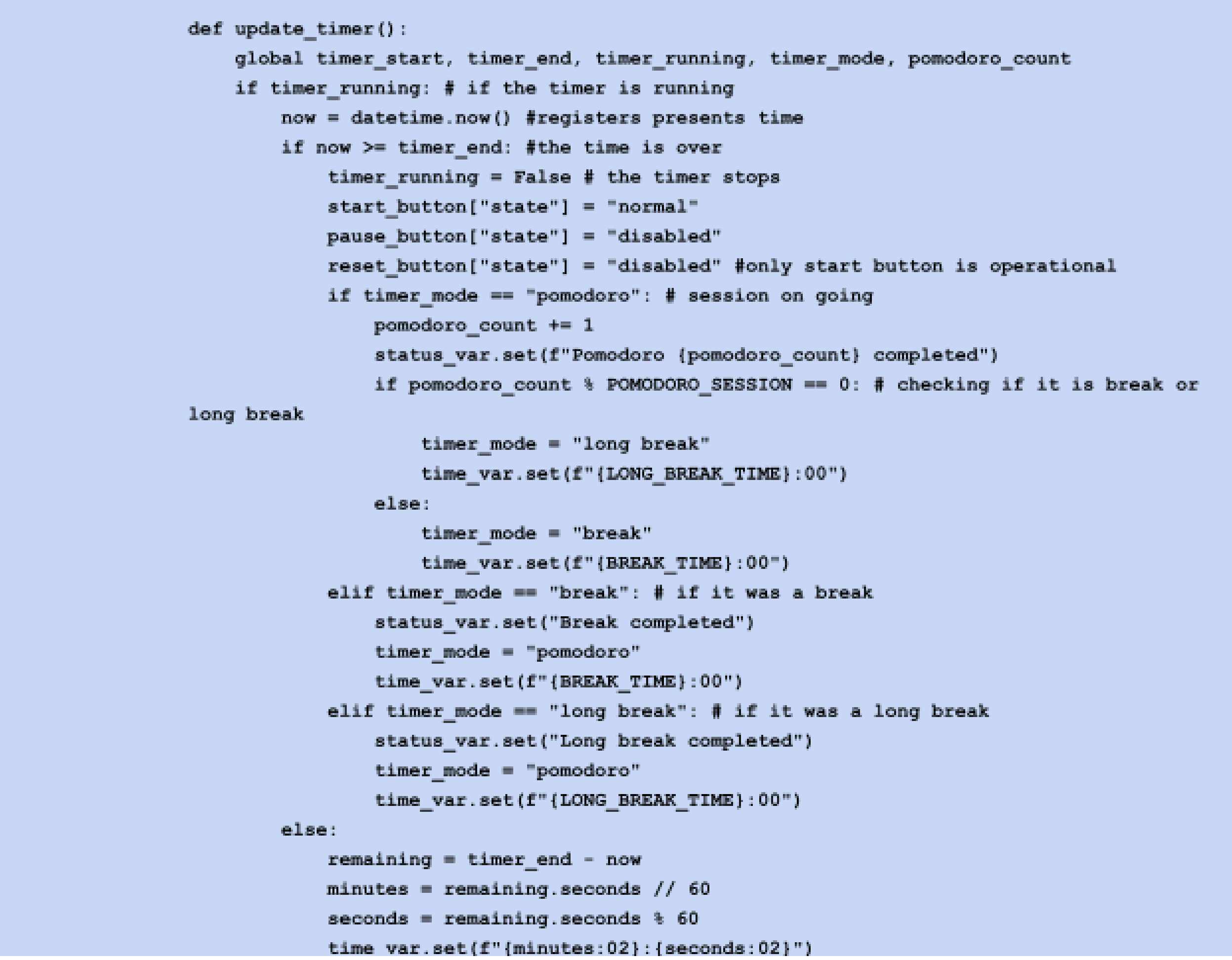
Modularity Check

The project comprises five distinct modules: main.py, pomodoro.py, goal.py, syllabus.py, and study.py. Each module serves a specific purpose within the project, contributing to its overall functionality and organization.

main.py module has the main window and frames of the user interface. **pomodoro.py** module serves the necessary features of the pomodoro timer. **goal.py** module serves the necessary features of the goal setter. **syllabus.py** module serves as the syllabus tracker features. **study.py** accumulates the modules and launch them all together.

If/else Condition to Switch statement

Python does not include a built-in switch case statement, consequently, there exists no alternative method to implement required conditions in the code besides using if/else statements.



```
def update_timer():
    global timer_start, timer_end, timer_running, timer_mode, pomodoro_count
    if timer_running: # if the timer is running
        now = datetime.now() #registers presents time
        if now >= timer_end: #the time is over
            timer_running = False # the timer stops
            start_button["state"] = "normal"
            pause_button["state"] = "disabled"
            reset_button["state"] = "disabled" #only start button is operational
            if timer_mode == "pomodoro": # session on going
                pomodoro_count += 1
                status_var.set(f"Pomodoro {pomodoro_count} completed")
                if pomodoro_count % POMODORO_SESSION == 0: # checking if it is break or
long break
                    timer_mode = "long break"
                    time_var.set(f"{LONG_BREAK_TIME}:00")
                else:
                    timer_mode = "break"
                    time_var.set(f"{BREAK_TIME}:00")
            elif timer_mode == "break": # if it was a break
                status_var.set("Break completed")
                timer_mode = "pomodoro"
                time_var.set(f"{BREAK_TIME}:00")
            elif timer_mode == "long break": # if it was a long break
                status_var.set("Long break completed")
                timer_mode = "pomodoro"
                time_var.set(f"{LONG_BREAK_TIME}:00")
        else:
            remaining = timer_end - now
            minutes = remaining.seconds // 60
            seconds = remaining.seconds % 60
            time_var.set(f"{minutes:02}:{seconds:02}")
```