Python Project Submission

System display/monitoring

01286121 Computer Programming

Software Engineering Program

By

66010988 Cusson Laohapatanawong

Python Project

System display/monitoring

Introduction

System display/monitoring allows users to monitor various aspects of their computer's performance in real-time. This capability provides valuable insights into the usage of critical system resources, helping users keep track of their system's health and performance. By visualizing metrics such as CPU usage, memory consumption, network activity, disk usage, temperature, and battery status, users can make informed decisions about resource optimization and troubleshooting potential issues.

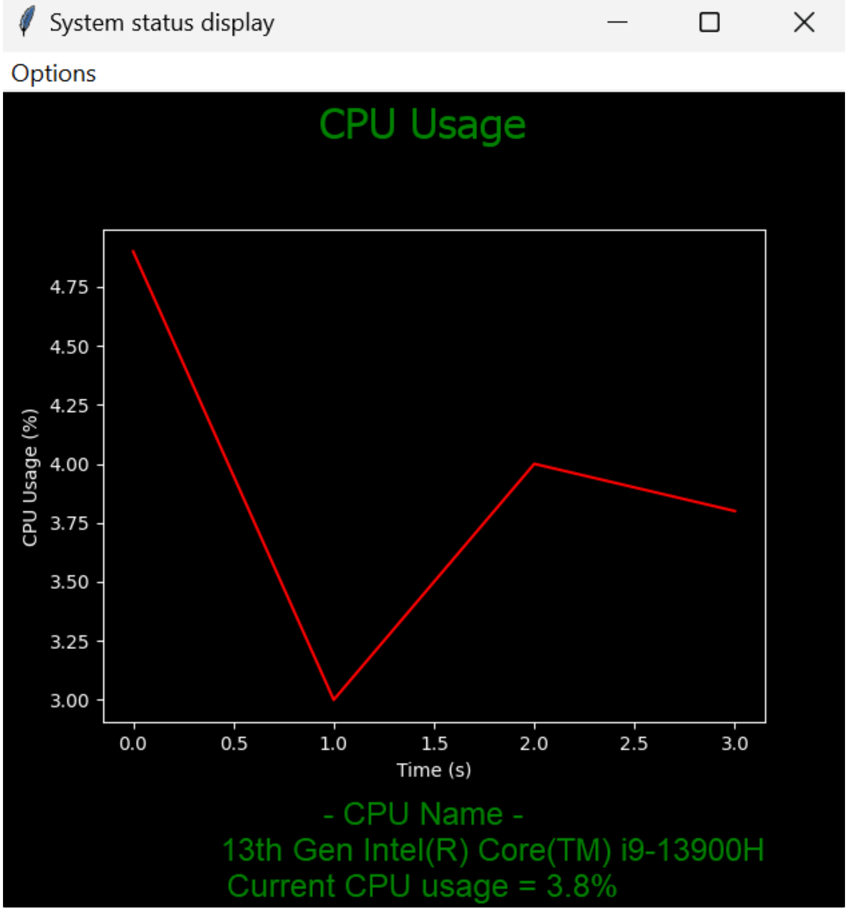
CPU Usage:

Real-time graphical representation of CPU usage percentage.

Enables users to identify periods of high or low CPU activity.

Main features:

- CPU name/type
- Real-time usage in percent



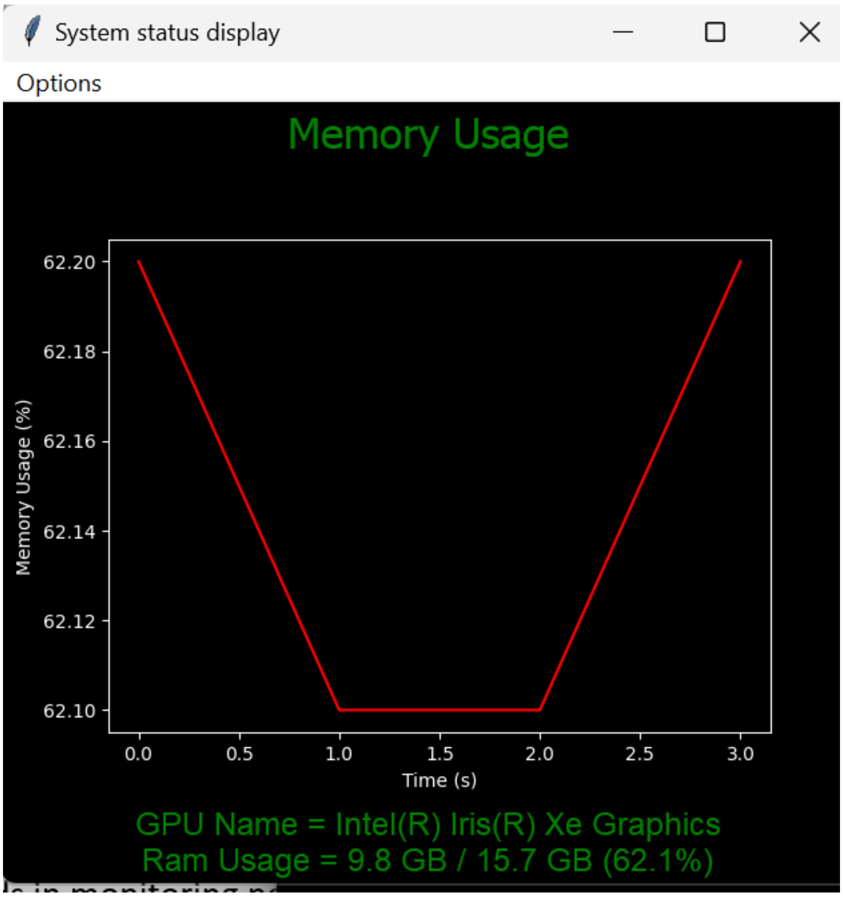
- CPU Name -
13th Gen Intel(R) Core(TM) i9-13900H
Current CPU usage = 3.8%

Memory usage:

Indicating both used and available memory. Allows users to gauge the efficiency of memory utilization by applications.

Main features:

- GPU name/type
- Used Ram / Total Ram
- Uses Percentage



GPU Name = Intel(R) Iris(R) Xe Graphics
Ram Usage = 9.8 GB / 15.7 GB (62.1%)

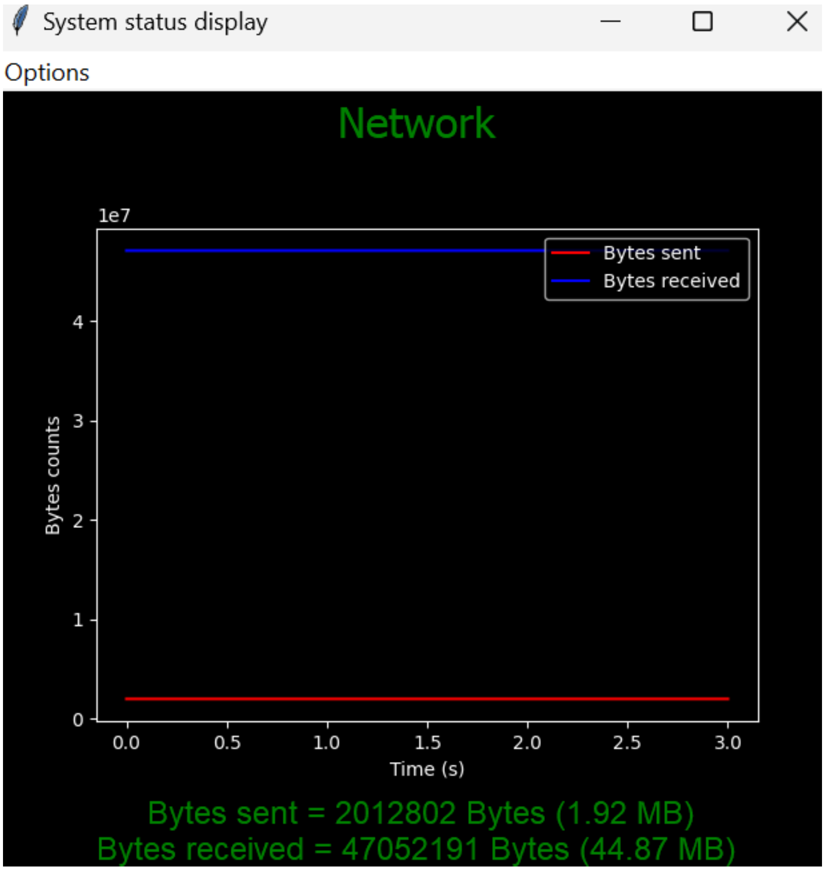
Network Activity:

Graphical representation of bytes sent and received over the network.

Aids in monitoring network performance and identifying potential bandwidth issues.

Main features:

- Byte sent in MB or GB
- Byte received in MB or GB



Bytes sent = 2012802 Bytes (1.92 MB)
Bytes received = 47052191 Bytes (44.87 MB)

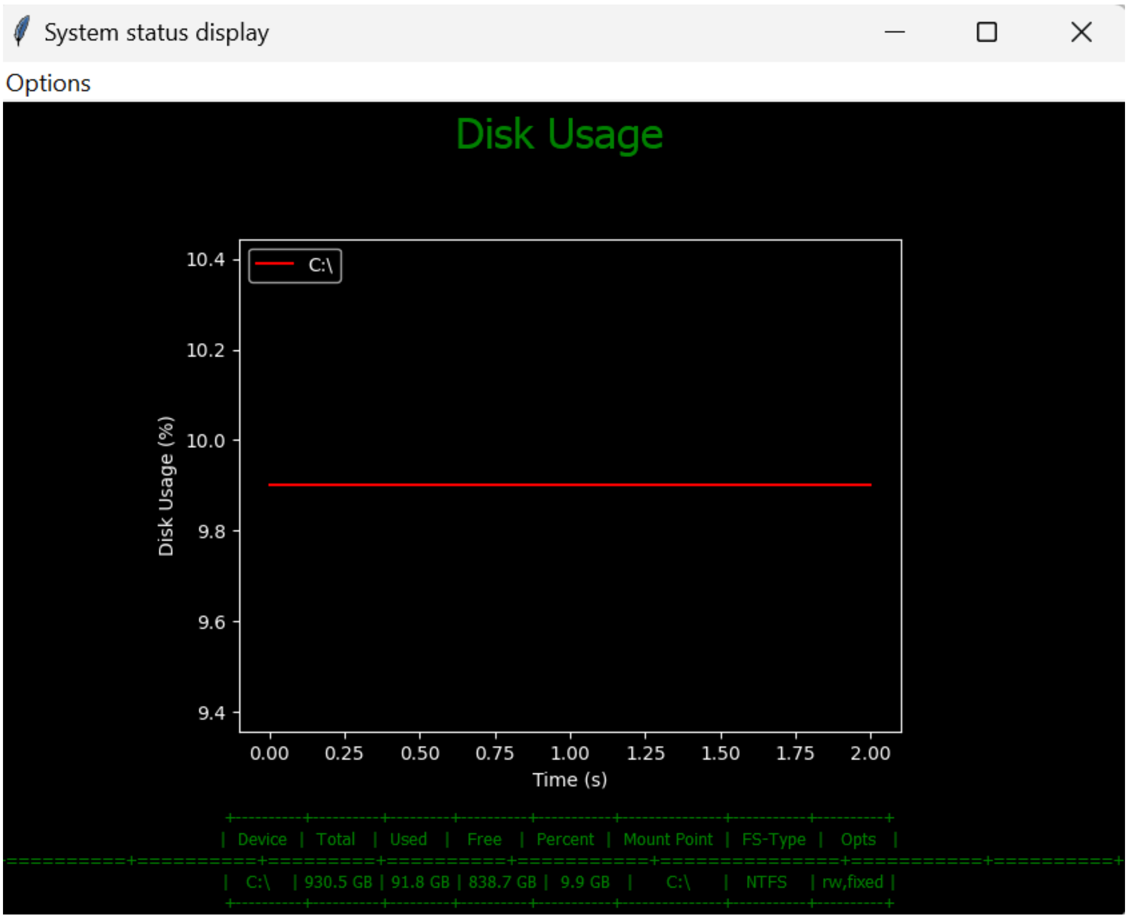
Disk Usage:

Visualizes the percentage of disk space currently in use.

Helps users manage storage capacity and avoid potential disk space shortages.

Main features:

- Table which shows all the Disk info



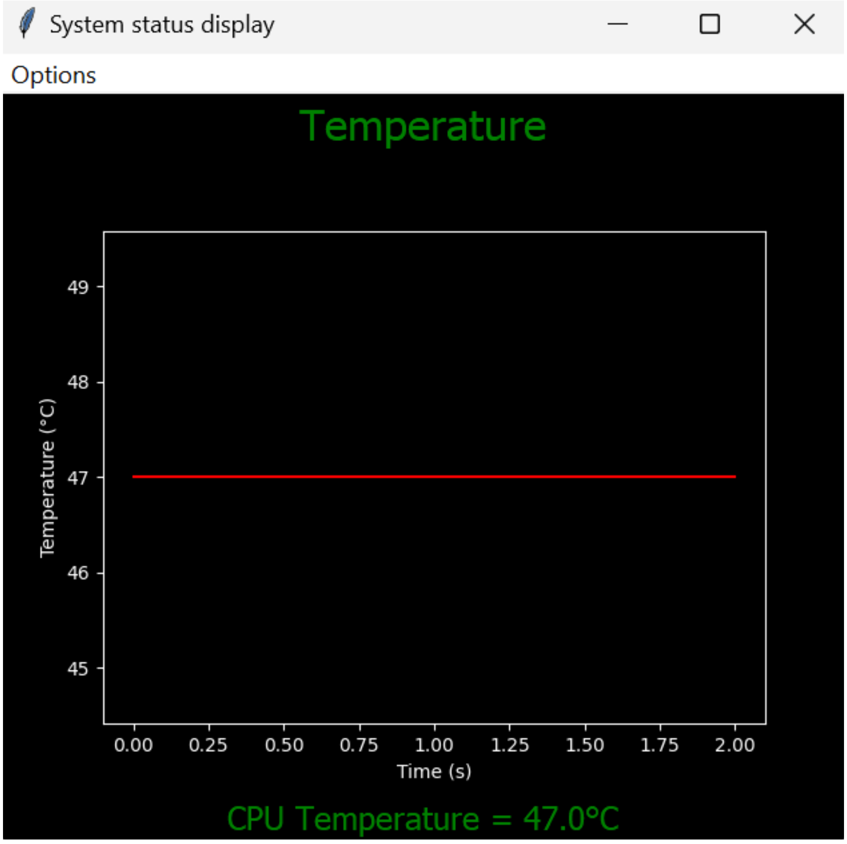
```
+----------+----------+---------+----------+----------+-------------+----------+----------+
|  Device  |  Total   |  Used   |   Free   | Percent  | Mount Point | FS-Type  |   Opts   |
+==========+==========+=========+==========+==========+=============+==========+==========+
|   C:\    | 930.5 GB | 91.8 GB | 838.7 GB |  9.9 GB  |     C:\     |   NTFS   | rw,fixed |
+----------+----------+---------+----------+----------+-------------+----------+----------+
```

Temperature Monitoring:

Displays real-time temperature information, crucial for systems with temperature-sensitive components. Assists in identifying overheating issues that may affect system stability.

Main features:

- Real-time graph and data of current core temperature



CPU Temperature = 47.0°C

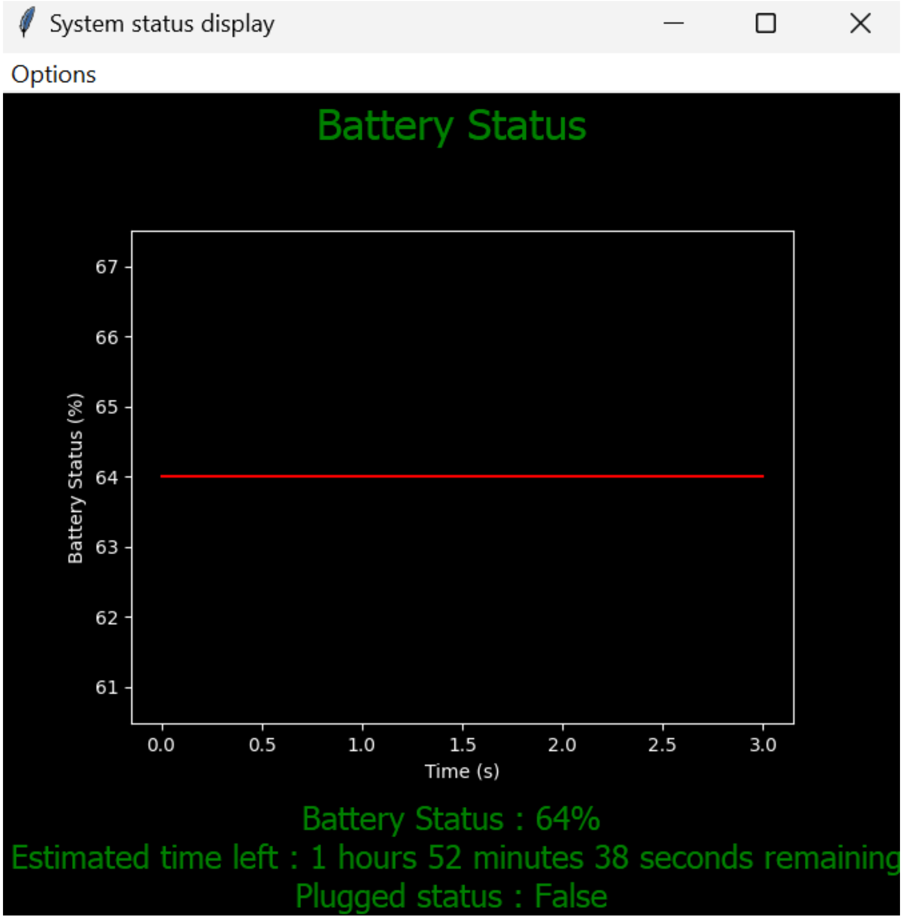
Battery Status (where applicable):

Monitors battery percentage for laptops and portable devices.

Provides insights into battery health and remaining charge.

Main features:

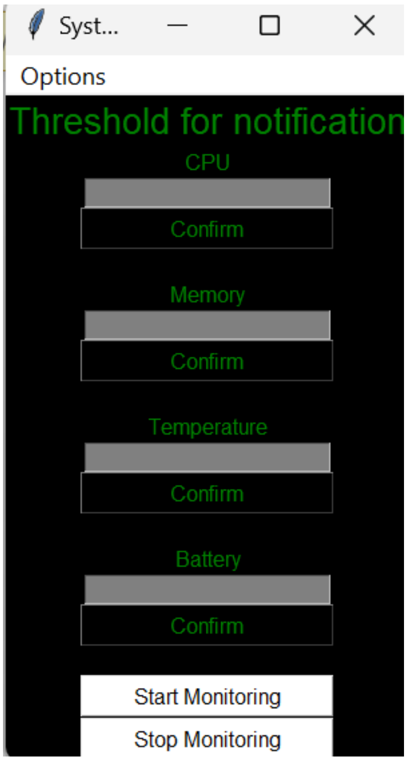
- Battery left in percentage
- Estimated time left
- Plugged status



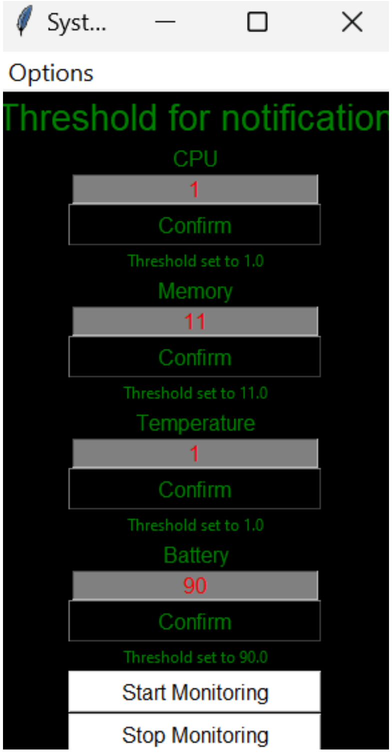
Battery Status : 64%
Estimated time left : 1 hours 52 minutes 38 seconds remaining
Plugged status : False

Notification:

User can set the threshold of each graph where if the usage reached/exceed the threshold, it will send a simple desktop notification to the user.



As an example, I will set the threshold so that it can reached it easily.



CPU
1

Threshold set to 1.0

Memory
11

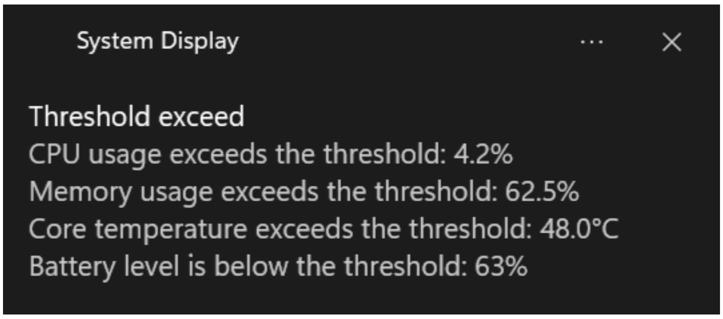
Threshold set to 11.0

Temperature
1

Threshold set to 1.0

Battery
90

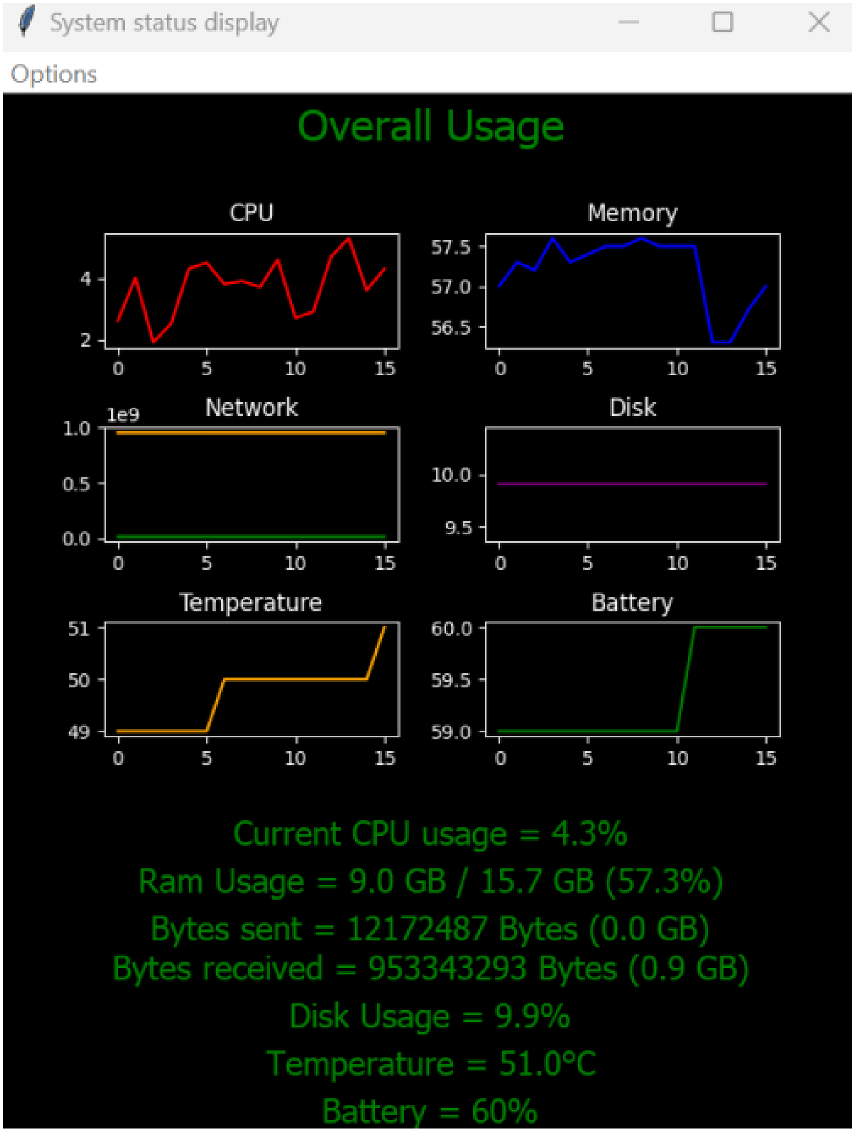
Threshold set to 90.0



Threshold exceed
CPU usage exceeds the threshold: 4.2%
Memory usage exceeds the threshold: 62.5%
Core temperature exceeds the threshold: 48.0°C
Battery level is below the threshold: 63%

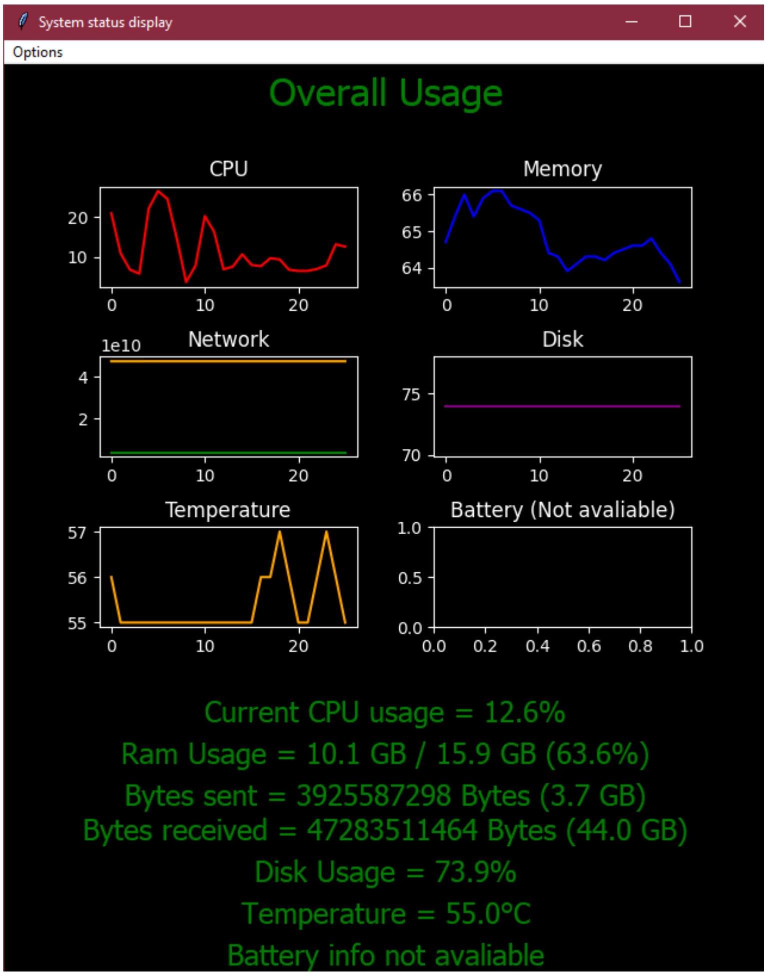
Overall page:

Briefly display all information and graph.



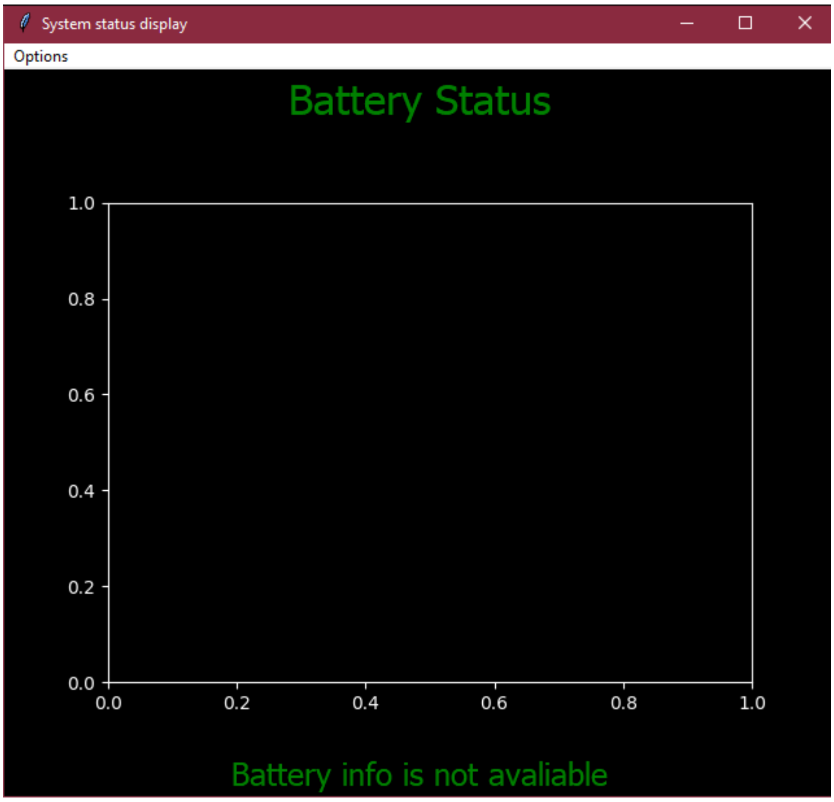
Current CPU usage = 4.3%
Ram Usage = 9.0 GB / 15.7 GB (57.3%)
Bytes sent = 12172487 Bytes (0.0 GB)
Bytes received = 953343293 Bytes (0.9 GB)
Disk Usage = 9.9%
Temperature = 51.0°C
Battery = 60%

When user is running a program on PC, the battery information will become unavailable.

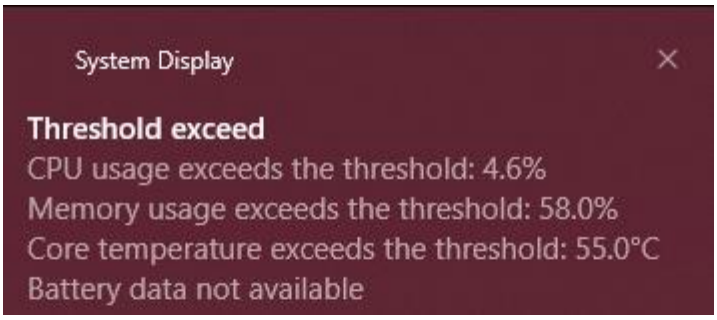


Overall Usage

Current CPU usage = 12.6%
Ram Usage = 10.1 GB / 15.9 GB (63.6%)
Bytes sent = 3925587298 Bytes (3.7 GB)
Bytes received = 47283511464 Bytes (44.0 GB)
Disk Usage = 73.9%
Temperature = 55.0°C
Battery info not avaliable



Battery info is not avaliable



Threshold exceed

CPU usage exceeds the threshold: 4.6%
Memory usage exceeds the threshold: 58.0%
Core temperature exceeds the threshold: 55.0°C
Battery data not available

Source code:

Github link : https://github.com/cussonspoon/All_Python_Year1/blob/main/Projectpy/System_Display.py

```
import tkinter as tk
from tkinter import Menu, Label, Canvas, Scrollbar, ttk
import abc
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.animation import FuncAnimation
import itertools
import psutil
from psutil import disk_partitions, disk_usage
from tabulate import tabulate
import psutil
import pandas as pd
import wmi
import re
import winotify
import GPUtil
import datetime
import time
```

```
plt.style.use('dark_background')
#----------------------------------------------------------------------------------------------------
---------------------------
#BASE CLASS
class SystemDisplayBase(abc.ABC):
    def __init__(self, window, frame, title):
        self.window = window
        self.frame = frame
        self.fig, self.ax = plt.subplots()
        self.title = title
        self.index = itertools.count()
        self.ani = None
        self.pc = wmi.WMI()

    def clear_frame(self):
        for widget in self.frame.winfo_children():
            widget.destroy()

    def set_header(self, title = ""):
        self.clear_frame()
        Label(self.frame, text=title, font=("Tahoma", 24) ,bg="Black", fg=
"Green").pack()
```

```
@abc.abstractmethod
def animate(self, i):
   pass

@abc.abstractmethod
def display(self):
   pass



def gigabytes_convert(self, byte):
   one_gb = 1073741824
   giga = byte / one_gb
   giga = "{0:.1f}".format(giga)
   return giga

def megabytes_convert(self, byte):
   one_mb = 1048576
   mega = byte/ one_mb
   mega = "{0:.2f}".format(mega)
   return mega

def details(self,device_name):
   for i in self.data:
      if i.device == device_name:
         return i
```

```
    def get_device_name(self):
        return [i.device for i in self.data]

    def disk_info(self,device_name):
        disk_info = {}
        try:
            usage = disk_usage(device_name)
            disk_info["Device"] = device_name
            disk_info["Total"] = f"{self.gigabytes_convert(usage.used + usage.free)}
GB"
            disk_info["Used"] = f"{self.gigabytes_convert(usage.used)} GB"
            disk_info["Free"] = f"{self.gigabytes_convert(usage.free)} GB"
            disk_info["Percent"] = f"{usage.percent} GB"
        except PermissionError:
            pass
        except FileNotFoundError:
            pass
        info = self.details(device_name)
        disk_info.update({"Device":info.device})
        disk_info["Mount Point"] = info.mountpoint
        disk_info["FS-Type"] = info.fstype
        disk_info["Opts"] = info.opts
        return disk_info

    def all_disk_info(self):
        return_all = []
        for i in self.get_device_name():
            return_all.append(self.disk_info(i))
        return return_all

#-------------------------------------------------------------------------------------------------------
---------------------------

class OverallDisplay(SystemDisplayBase):
    def __init__(self, window, frame, fig):
        super().__init__(window, frame, "Overall Usage")
        self.x_vals = []
        self.cpu_y_vals = []

        self.memory_y_vals = []

        self.network_y_vals1 = []
        self.network_y_vals2 = []

        self.disk_y_vals = []

        self.temp_y_vals = []

        self.battery_y_vals = []

    def create_subplots(self):
        self.fig , self.ax = plt.subplots(3, 2)
        self.fig.subplots_adjust(hspace=0.7, wspace=0.3)
        self.ax[0, 0].set_title("CPU")
        self.ax[0, 1].set_title("Memory")
        self.ax[1, 0].set_title("Network")
        self.ax[1, 1].set_title("Disk")
        self.ax[2, 0].set_title("Temperature")
        self.ax[2, 1].set_title("Battery")
    def animate(self, i):
        # Update CPU graph
        self.x_vals.append(next(self.index))
        self.cpu_y_vals.append(psutil.cpu_percent())
        self.ax[0, 0].clear()
        self.ax[0, 0].plot(self.x_vals, self.cpu_y_vals, label="CPU Usage", color="red")
        self.ax[0, 0].set_title("CPU")

        # Update Memory graph
        self.memory_y_vals.append(psutil.virtual_memory().percent)
        self.ax[0, 1].clear()
        self.ax[0, 1].plot(self.x_vals, self.memory_y_vals, label="Memory Usage", color="blue")
        self.ax[0, 1].set_title("Memory")

        # Update Network graph
        self.network_y_vals1.append(psutil.net_io_counters().bytes_sent)
        self.network_y_vals2.append(psutil.net_io_counters().bytes_recv)
        self.ax[1, 0].clear()
        self.ax[1, 0].plot(self.x_vals, self.network_y_vals1, label="Bytes sent", color="green")
        self.ax[1, 0].plot(self.x_vals, self.network_y_vals2, label="Bytes received", color="orange")
        self.ax[1, 0].set_title("Network")

        # Update Disk graph
        disk_usage = psutil.disk_usage('/')
        self.disk_y_vals.append(disk_usage.percent)
        self.ax[1, 1].clear()
        self.ax[1, 1].plot(self.x_vals, self.disk_y_vals, label="Disk Usage", color="purple")
        self.ax[1, 1].set_title("Disk")

        #Update the Temperature graph
        gpu = GPUtil.getGPUs()[0]
        temp = gpu.temperature
        self.temp_y_vals.append(temp)
        self.ax[2, 0].clear()
        self.ax[2, 0].plot(self.x_vals, self.temp_y_vals, label= "Temperature (°C)", color = "orange")
        self.ax[2, 0].set_title("Temperature")

        #Update the Battery graph
        try:
            battery = psutil.sensors_battery()
            battery = battery.percent
            self.battery_y_vals.append(battery)
            self.ax[2, 1].clear()
```

```
       self.ax[2, 1].plot(self.x_vals, self.battery_y_vals, label= "Battery (%)",
color = "green")
       self.ax[2, 1].set_title("Battery")
    except:
        self.ax[2, 1].set_title("Battery (Not avaliable)")

    # Update CPU and Memory info labels
    cpu_info = f"Current CPU usage = {self.cpu_y_vals[-1]}%"
    memory_info = f"Ram Usage =
{self.gigabytes_convert(psutil.virtual_memory().used)} GB /
{self.gigabytes_convert(psutil.virtual_memory().total)} GB
({psutil.virtual_memory().percent}%)"

    # Update network info label
    sent = psutil.net_io_counters().bytes_sent
    rec = psutil.net_io_counters().bytes_recv
    network_info = f"Bytes sent = {sent} Bytes ({self.gigabytes_convert(sent)}
GB)\nBytes received = {rec} Bytes ({self.gigabytes_convert(rec)} GB)"

    # Update disk info label
    disk_info = f"Disk Usage = {disk_usage.percent}%"

    #Update temperature info label
    temp_info = f"Temperature = {temp}°C"

    #Update Battery info label
    try:
       if battery == None:
```

```
                raise ValueError
            else:
                batt_info = f"Battery = {battery}%"
        except ValueError:
            batt_info = f"Battery info not avaliable"

        #Update the info of all Labels
        self.cpu_info_label.config(text=cpu_info)
        self.memory_info_label.config(text=memory_info)
        self.network_info_label.config(text=network_info)
        self.disk_info_label.config(text=disk_info)
        self.temp_info_label.config(text= temp_info)
        self.batt_info_label.config(text= batt_info)

    def display(self):
        super().display()
        self.clear_frame()
        self.set_header("Overall Usage")
        self.create_subplots()
        index = itertools.count()
        self.ani = FuncAnimation(self.fig, self.animate, frames=index,
interval=1000)
        canvas_overall = FigureCanvasTkAgg(self.fig, master=self.frame)
        canvas_widget_overall = canvas_overall.get_tk_widget()
        canvas_widget_overall.pack()
        self.cpu_info_label = Label(self.frame, text="", font=("Tahoma", 18),
bg="Black", fg="Green")
```

```
        self.cpu_info_label.pack()
        self.memory_info_label = Label(self.frame, text="", font=("Tahoma", 18),
bg="Black", fg="Green")
        self.memory_info_label.pack()
        self.network_info_label = Label(self.frame, text="", font=("Tahoma", 18),
bg="Black", fg="Green")
        self.network_info_label.pack()
        self.disk_info_label = Label(self.frame, text="", font=("Tahoma", 18),
bg="Black", fg="Green")
        self.disk_info_label.pack()
        self.temp_info_label = Label(self.frame, text="", font=("Tahoma", 18),
bg="Black", fg="Green")
        self.temp_info_label.pack()
        self.batt_info_label = Label(self.frame, text="", font=("Tahoma", 18),
bg="Black", fg="Green")
        self.batt_info_label.pack()


class CPUDisplay(SystemDisplayBase):
    def __init__(self, window, frame):
        super().__init__(window, frame, "CPU Usage")
        self.cpu_info_label = Label(self.frame, text="", font=("Tahoma", 18)
,bg="Black", fg= "Green")
        self.cpu_info_label.pack()
        self.cpu_x_vals = []
        self.cpu_y_vals = []


    def animate(self, i):
        self.cpu_x_vals.append(next(self.index))
        self.cpu_y_vals.append(psutil.cpu_percent())
        self.ax.clear()
        self.ax.plot(self.cpu_x_vals, self.cpu_y_vals,color= "red")
        self.ax.set_xlabel("Time (s)")
        self.ax.set_ylabel("CPU Usage (%)")
        self.cpu_info_label.config(text=f"- CPU Name -
\n\t{self.pc.Win32_Processor()[0].name}\nCurrent CPU usage =
{self.cpu_y_vals[-1]}%")

    def display(self):
        self.clear_frame()
        self.set_header("CPU Usage")
        self.ani = FuncAnimation(self.fig, self.animate, frames=100, interval=1000)
        canvas_cpu = FigureCanvasTkAgg(self.fig, master=self.frame)
        canvas_widget_cpu = canvas_cpu.get_tk_widget()
        canvas_widget_cpu.pack()
        self.cpu_info_label = Label(self.frame, text="", font = (hasattr, 18),
bg="Black", fg= "Green")
        self.cpu_info_label.pack()

#----------------------------------------------------------------------------------------------------
---------------------------

class MemoryDisplay(SystemDisplayBase):
    def __init__(self, window, frame, fig):
        super().__init__(window, frame,"Memory Usage")
        self.memory_info_label = Label(self.frame, text="", font=("Tahoma",
18),bg="Black", fg= "Green")
```

```
        self.memory_info_label.pack()

        self.memory_x_vals = []

        self.memory_y_vals = []



    def animate(self, i):

        self.memory_x_vals.append(next(self.index))

        self.memory_y_vals.append(psutil.virtual_memory().percent)

        self.ax.clear()

        self.ax.plot(self.memory_x_vals, self.memory_y_vals,color= "red")

        self.ax.set_xlabel("Time (s)")

        self.ax.set_ylabel("Memory Usage (%)")

        self.memory_info_label.config(text=f"GPU Name =
{self.pc.Win32_VideoController()[0].name}\nRam Usage =
{self.gigabytes_convert(psutil.virtual_memory().used)} GB /
{self.gigabytes_convert(psutil.virtual_memory().total)} GB
({psutil.virtual_memory().percent}%)")


    def display(self):

        self.clear_frame()

        self.set_header("Memory Usage")

        self.ani = FuncAnimation(self.fig, self.animate, frames=100, interval=1000)

        canvas_mem = FigureCanvasTkAgg(self.fig, master=self.frame)

        canvas_widget_mem = canvas_mem.get_tk_widget()

        canvas_widget_mem.pack()

        self.memory_info_label = Label(self.frame, text="", font = (hasattr,
18),bg="Black", fg= "Green" )

        self.memory_info_label.pack()
```

```
#----------------------------------------------------------------------------------------------------
---------------------------

class NetworkDisplay(SystemDisplayBase):
    def __init__(self, window, frame, fig):
        super().__init__(window, frame,"Network")
        self.network_info_label = Label(self.frame, text="", font=("Tahoma",
18),bg="Black", fg= "Green")
        self.network_info_label.pack()
        self.network_x_vals = []
        self.network_y_vals1 = []
        self.network_y_vals2 = []

    def animate(self, i):
        self.network_x_vals.append(next(self.index))
        self.network_y_vals1.append(psutil.net_io_counters().bytes_sent)
        self.network_y_vals2.append(psutil.net_io_counters().bytes_recv)
        self.ax.clear()
        self.ax.plot(self.network_x_vals, self.network_y_vals1, label="Bytes
sent",color= "red")
        self.ax.plot(self.network_x_vals, self.network_y_vals2, label="Bytes
received", color = "Blue")
        self.ax.set_xlabel("Time (s)")
        self.ax.set_ylabel("Bytes counts")
        self.ax.legend(loc = "upper right")
        sent = psutil.net_io_counters().bytes_sent
        rec = psutil.net_io_counters().bytes_recv
        if sent > 1073741824/10 and rec >1073741824/10:
            self.network_info_label.config(text = f" Bytes sent = {sent} Bytes ({self.gigabytes_convert(sent)} GB)\nBytes received = {rec} Bytes ({self.gigabytes_convert(rec)} GB)")
        elif sent > 1048576/10 and rec > 1048576/10:
            self.network_info_label.config(text = f" Bytes sent = {sent} Bytes ({self.megabytes_convert(sent)} MB)\nBytes received = {rec} Bytes ({self.megabytes_convert(rec)} MB)")


    def display(self):
        super().display()
        self.clear_frame()
        self.set_header("Network")
        index = itertools.count()
        self.ani = FuncAnimation(self.fig, self.animate, frames = index, interval = 1000)
        canvas_network = FigureCanvasTkAgg(self.fig, master=self.frame)
        canvas_widgit_memory = canvas_network.get_tk_widget()
        canvas_widgit_memory.pack()
        self.network_info_label = Label(self.frame, text ="", font = (hasattr, 18),bg="Black", fg= "Green")
        self.network_info_label.pack()


#----------------------------------------------------------------------------------------------------------------------------


class DiskDisplay(SystemDisplayBase):
    def __init__(self, window, frame, fig):
        super().__init__(window, frame,"Disk Usage")
        self.disk_info_label = Label(self.frame, text="", font=("Tahoma",
12),bg="Black", fg= "Green")
        self.disk_info_label.pack()
        self.disk_x_vals = []
        self.disk_y_vals = []
        self.data = disk_partitions(all = False)

    def animate(self, i):
        self.disk_x_vals.append(next(self.index))
        device_names = self.get_device_name()

        while len(self.disk_y_vals) < len(device_names):
            self.disk_y_vals.append([])

        for i, name in enumerate(device_names):
            try:
                usage = disk_usage(name).percent
                self.disk_y_vals[i].append(usage)
            except (PermissionError, FileNotFoundError):
                self.disk_y_vals[i].append(None)

        self.ax.clear()
        self.ax.set_xlabel("Time (s)")
        self.ax.set_ylabel("Disk Usage (%)")
        for i, name in enumerate(device_names):
            self.ax.plot(self.disk_x_vals, self.disk_y_vals[i], label=name,color=
"Red")

        self.ax.legend(loc='upper left')



        disk_info = self.all_disk_info()



        for info in disk_info:
            info['Device'] = f" {info['Device']}"
            info['Total'] = f" {info['Total']}"
            info['Used'] = f" {info['Used']}"
            info['Free'] = f" {info['Free']}"
            info['Percent'] = f" {info['Percent']}"

        disk_info_text = tabulate(
            disk_info,
            headers="keys",
            tablefmt="grid",
            showindex=False,
            numalign="center",
            stralign="left",
            colalign=("center",) * len(disk_info[0].keys()),
        )
        self.disk_info_label.config(text=disk_info_text, font=("Tahoma",
10),bg="Black", fg= "Green")


    def display(self):
        super().display()
        self.clear_frame()
        self.set_header("Disk Usage")


        index = itertools.count()
        self.ani = FuncAnimation(self.fig, self.animate, frames=index,
interval=1000)
        canvas_disk = FigureCanvasTkAgg(self.fig, master=self.frame)
        canvas_widget_disk = canvas_disk.get_tk_widget()
        canvas_widget_disk.pack()


        self.disk_info_label = Label(self.frame, text="", font=("Tahoma",
12),bg="Black", fg= "Green")
        self.disk_info_label.pack()


#-----------------------------------------------------------------------------------------------------------
---------------------------


class ProcessDisplay(SystemDisplayBase):
    def __init__(self, window, frame):
        super().__init__(window, frame, "Process Monitor")
        self.process_table_label = Label(self.frame, text="", font=("Tahoma",
12),bg="Black", fg= "Green")
```

```
        self.process_table_label.pack()
        self.process_x_vals = []
        self.process_y_vals = []
        self.selected_process_pid = None  # Store the PID of the selected process

        self.fig, self.ax = plt.subplots()

    def animate(self, i):
            self.process_x_vals.append(next(self.index))

            # Check if a process is selected
            if self.selected_process_pid is not None:
                # Fetch memory usage for the selected process
                selected_process = psutil.Process(self.selected_process_pid)
                memory_percent = selected_process.memory_percent()
                self.process_y_vals.append(memory_percent)
                selected_process = psutil.Process(self.selected_process_pid)
                selected_process_name = selected_process.name()
                selected_process_memory_usage =
selected_process.memory_percent()
                selected_process_info = f"Overall memory usage :
{psutil.virtual_memory().percent}%\n\nSelected Process:
{selected_process_name}\n\nMemory Usage:
{selected_process_memory_usage:.2f}% / {psutil.virtual_memory().percent}%"

                # Create a label to display the selected process info
                self.process_table_label.config(text=selected_process_info,
font=("Tahoma", 15),bg="Black", fg= "Green")
```

```
            self.process_table_label.pack()
        else:
            # Display overall memory usage if no process is selected
            memory_percent = psutil.virtual_memory().percent
            self.process_y_vals.append(memory_percent)
            self.process_table_label.config(text=f"Overall memory usage :
{memory_percent}% ", font=("Tahoma", 15),bg="Black", fg= "Green")
            self.process_table_label.pack()

        # Update the graph
        self.ax.clear()
        self.ax.plot(self.process_x_vals, self.process_y_vals,color= "red")
        self.ax.set_xlabel("Time (s)")
        self.ax.set_ylabel("Memory Usage (%)")

    def display(self):
        super().display()
        self.clear_frame()
        self.set_header("Process Monitor")

        index = itertools.count()
        self.ani = FuncAnimation(self.fig, self.animate, frames=index,
interval=1000)
        canvas_process = FigureCanvasTkAgg(self.fig, master=self.frame)
        canvas_widget_process = canvas_process.get_tk_widget()
        canvas_widget_process.pack()
        self.process_table_label = Label(self.frame, text="", font=("Tahoma", 12),bg="Black", fg= "Green")
        process_treeview = self.create_process_treeview(self.frame, self.get_top_memory_processes())
        process_treeview.bind("<Double-1>", self.on_process_row_double_click)

    def create_process_treeview(self, frame, process_data):
        # Create a frame to hold the Treeview and the scrollbar
        table_frame = tk.Frame(frame)
        table_frame.pack(side=tk.LEFT, padx=10, pady=10)

        # Create the Treeview widget inside the table frame
        treeview = ttk.Treeview(table_frame, columns=("PID", "Name", "Status"), show="headings")
        treeview.heading("PID", text="PID")
        treeview.heading("Name", text="Name")
        treeview.heading("Status", text="Status")

        for data in process_data:
            treeview.insert("", "end", values=data)

        treeview.pack(side=tk.LEFT)  # Pack the Treeview to the left within the table frame

        # Add a vertical scrollbar to the right of the table frame
        y_scrollbar = ttk.Scrollbar(table_frame, orient=tk.VERTICAL, command=treeview.yview)
        y_scrollbar.pack(side=tk.RIGHT, fill=tk.Y)
        # Configure the Treeview to use the scrollbar
        treeview.configure(yscrollcommand=y_scrollbar.set)

        return treeview

    def display_process_graph(self):
        self.process_x_vals = []
        self.process_y_vals = []

        self.clear_frame()
        self.set_header("Process Monitor")

        index = itertools.count()
        self.ani = FuncAnimation(self.fig, self.animate, frames=index, interval=1000)
        canvas_process = FigureCanvasTkAgg(self.fig, master=self.frame)
        canvas_widget_process = canvas_process.get_tk_widget()
        canvas_widget_process.pack()
        self.process_table_label = Label(self.frame, text="", font=("Tahoma", 12),bg="Black", fg= "Green")
        self.process_table_label.pack()

        # Bind a double-click event to the process table to select a process
        process_treeview = self.create_process_treeview(self.frame, self.get_top_memory_processes())
        process_treeview.bind("<Double-1>", self.on_process_row_double_click)
        # Display selected process name and memory usage
        self.process_table_label = Label(self.frame, text ="", font = (hasattr,
18),bg="Black", fg= "Green")
        self.process_table_label.pack()

    def on_process_row_double_click(self, event):
        item = event.widget.selection()[0]
        process_pid_str = event.widget.item(item, "values")[0]

        # Use regular expressions to extract the numeric part
        process_pid_match = re.search(r'(\d+)', process_pid_str)
        if process_pid_match:
            process_pid = int(process_pid_match.group(1))
            self.select_process(process_pid)

    def clear_graph_data(self):
        self.process_x_vals = []
        self.process_y_vals = []

    def select_process(self, process_pid):
        self.selected_process_pid = process_pid
        self.clear_graph_data()
        self.display_process_graph()

    def get_top_memory_processes(self):
        processes = list(psutil.process_iter(attrs=['pid', 'name', 'memory_percent']))
        return sorted(processes, key=lambda x: x.info['memory_percent'], reverse=True)

#----------------------------------------------------------------------------------------------------------------------------

class TemperatureDisplay(SystemDisplayBase):
    def __init__(self, window, frame, fig):
        super().__init__(window, frame, "Temperature")
        self.temperature_info_label = Label(self.frame, text="", font=("Tahoma", 18),bg="Black", fg= "Green")
        self.temperature_info_label.pack()
        self.temperature_x_vals = []
        self.temperature_y_vals = []

    def animate(self, i):
        self.temperature_x_vals.append(next(self.index))
        gpu = GPUtil.getGPUs()[0]
        temp = gpu.temperature
        self.temperature_y_vals.append(temp)
        self.ax.clear()
        self.ax.plot(self.temperature_x_vals, self.temperature_y_vals,color= "red")
        self.ax.set_xlabel("Time (s)")
        self.ax.set_ylabel("Temperature (°C)")
        self.temperature_info_label.config(text=f"CPU Temperature = {temp}°C")
    def display(self):
        super().display()
        self.clear_frame()
        self.set_header("Temperature")
        index = itertools.count()
        self.ani = FuncAnimation(self.fig, self.animate, frames=index,
interval=1000)
        canvas_temp = FigureCanvasTkAgg(self.fig, master=self.frame)
        canvas_widget_temp = canvas_temp.get_tk_widget()
        canvas_widget_temp.pack()
        self.temperature_info_label = Label(self.frame, text="", font=("Tahoma",
18),bg="Black", fg= "Green")
        self.temperature_info_label.pack()


#----------------------------------------------------------------------------------------------------
---------------------------
class BatteryDisplay(SystemDisplayBase):
    def __init__(self, window, frame, fig):
        super().__init__(window, frame, "Battery Status")
        self.battery_info_label = Label(self.frame, text="", font=("Tahoma",
18),bg="Black", fg= "Green")
        self.battery_info_label.pack()
        self.battery_x_vals = []
        self.battery_y_vals = []


    def animate(self, i):
        try:
            self.battery_x_vals.append(next(self.index))
            battery = psutil.sensors_battery()
            battery_percent = battery.percent
            battery_left = battery.secsleft
            self.battery_y_vals.append(battery_percent)
            self.ax.clear()
            self.ax.plot(self.battery_x_vals, self.battery_y_vals,color= "red")
            self.ax.set_xlabel("Time (s)")
            self.ax.set_ylabel("Battery Status (%)")
            self.battery_info_label.config(text=f"Battery Status : {battery_percent}%\n Estimated time left : {self.convert_to_hour(battery_left)}\nPlugged status : {battery.power_plugged}")
        except:
            self.battery_info_label.config(text = "Battery info is not avaliable")


    def convert_to_hour(self, sec):
        if sec is not None and sec > 0:
            time_delta = datetime.timedelta(seconds=sec)
            hours, remainder = divmod(time_delta.seconds, 3600)
            minutes, seconds = divmod(remainder, 60)
            return f"{hours} hours {minutes} minutes {seconds} seconds remaining"
        else:
            return "N/A"


    def display(self):
        super().display()
        self.clear_frame()
        self.set_header("Battery Status")
        index = itertools.count()
        self.ani = FuncAnimation(self.fig, self.animate, frames=index,
interval=1000)
        canvas_battery = FigureCanvasTkAgg(self.fig, master=self.frame)
        canvas_widget_battery = canvas_battery.get_tk_widget()
        canvas_widget_battery.pack()
        self.battery_info_label = Label(self.frame, text="", font=("Tahoma",
18),bg="Black", fg= "Green")
        self.battery_info_label.pack()


#----------------------------------------------------------------------------------------------------
---------------------------
class Notification(SystemDisplayBase):
    def animate(self, i):
        pass
    def __init__(self, window, frame):
        super().__init__(window, frame, "Notification")
        self.monitoring_started = False
        self.after_id = None

    def start_monitoring(self):
        if not self.monitoring_started:
            self.monitoring_started = True
            self.monitor_system()
    def stop_monitoring(self):
        self.monitoring_started = False
        if self.after_id:
            self.frame.after_cancel(self.after_id)


    def check_value(self, value, label):
        try:
            value = float(value)
            label.config(text=f"Threshold set to {value}")
        except ValueError:
            label.config(text="Invalid input. Please enter valid numeric values.")
    def shownoti(self, mssg):
        noti = winotify.Notification(app_id = "System Display", title = "Threshold
exceed", msg = mssg, duration  = "long")
        noti.show()
    def monitor_system(self):
        if self.monitoring_started:
            cpu_percent = psutil.cpu_percent()
            memory_percent = psutil.virtual_memory().percent
            gpu = GPUtil.getGPUs()[0]
            temperatures = gpu.temperature
            msg = ""
            try:
                battery = psutil.sensors_battery()
                battery = battery.percent
            except:
```

```
            battery = None

        if cpu_percent > float(self.cpu_thold.get()):
            msg += f"CPU usage exceeds the threshold: {cpu_percent}%\n"
        elif cpu_percent == float(self.cpu_thold.get()):
            msg += f"CPU usage is at the threshold: {cpu_percent}%\n"

        if memory_percent > float(self.mem_thold.get()):
            msg += f"Memory usage exceeds the threshold:
{memory_percent}%\n"
        elif memory_percent == float(self.mem_thold.get()):
            msg += f"Memory usage is at the threshold: {memory_percent}%\n"
        if temperatures is not None:
            if temperatures > float(self.temp_thold.get()):
                msg += f"Core temperature exceeds the threshold:
{temperatures}°C\n"
            elif temperatures == float(self.temp_thold.get()):
                msg += f"Core temperature is at the threshold: {temperatures}°C\n"
        else:
            msg += "Temperature data not available\n"

        if battery is not None:
            if battery < float(self.batt_thold.get()):
                msg += f"Battery level is below the threshold: {battery}%\n"
        else:
            msg += "Battery data not available\n"
            if msg:
                self.shownoti(msg)

            self.after_id = self.frame.after(10000, self.monitor_system)



    def display(self):
        super().display()
        self.clear_frame()

        self.cpu_thold = tk.StringVar()
        self.mem_thold = tk.StringVar()
        self.temp_thold = tk.StringVar()
        self.batt_thold = tk.StringVar()

        self.title = tk.Label(self.frame, text="Threshold for notification",
font=("Arial", 20), justify="center", bg="black", fg="green")
        self.cpu_label = tk.Label(self.frame, text="CPU", font=("Arial", 12),
justify='center', bg="black", fg="green")
        self.cpu_label2 = tk.Label(self.frame, bg="black", fg="green")
        self.cpu_entry = tk.Entry(self.frame, width=20,
textvariable=self.cpu_thold, font=('Arial', 12), justify='center', bg="grey",
fg="red")
        self.confirm_button_cpu = tk.Button(self.frame, width=20,
text="Confirm", font=("Arial", 12), justify="center", bg="black", fg="green",
command=lambda: self.check_value(self.cpu_thold.get(), self.cpu_label2))
        self.mem_label = tk.Label(self.frame, text="Memory", font=("Arial", 12),
justify='center', bg="black", fg="green")
```

```
        self.mem_label2 = tk.Label(self.frame, bg="black", fg="green")

        self.mem_entry = tk.Entry(self.frame, width=20, textvariable=self.mem_thold, font=('Arial', 12), justify='center', bg="grey", fg="red")

        self.confirm_button_mem = tk.Button(self.frame, width=20, text="Confirm", font=("Arial", 12), justify="center", bg="black", fg="green", command=lambda: self.check_value(self.mem_thold.get(), self.mem_label2))

        self.temp_label = tk.Label(self.frame, text="Temperature", font=("Arial", 12), justify='center', bg="black", fg="green")

        self.temp_label2 = tk.Label(self.frame, bg="black", fg="green")

        self.temp_entry = tk.Entry(self.frame, width=20, textvariable=self.temp_thold, font=('Arial', 12), justify='center', bg="grey", fg="red")

        self.confirm_button_temp = tk.Button(self.frame, width=20, text="Confirm", font=("Arial", 12), justify="center", bg="black", fg="green", command=lambda: self.check_value(self.temp_thold.get(), self.temp_label2))

        self.batt_label = tk.Label(self.frame, text="Battery", font=("Arial", 12), justify='center', bg="black", fg="green")

        self.batt_label2 = tk.Label(self.frame, bg="black", fg="green")

        self.batt_entry = tk.Entry(self.frame, width=20, textvariable=self.batt_thold, font=('Arial', 12), justify='center', bg="grey", fg="red")

        self.confirm_button_batt = tk.Button(self.frame, width=20, text="Confirm", font=("Arial", 12), justify="center", bg="black", fg="green", command=lambda: self.check_value(self.batt_thold.get(), self.batt_label2))


        # Pack the widgets

        self.title.pack()

        self.cpu_label.pack()

        self.cpu_entry.pack()
        self.confirm_button_cpu.pack()
        self.cpu_label2.pack()
        self.mem_label.pack()
        self.mem_entry.pack()
        self.confirm_button_mem.pack()
        self.mem_label2.pack()
        self.temp_label.pack()
        self.temp_entry.pack()
        self.confirm_button_temp.pack()
        self.temp_label2.pack()
        self.batt_label.pack()
        self.batt_entry.pack()
        self.confirm_button_batt.pack()
        self.batt_label2.pack()
        self.start_button = tk.Button(self.frame, width=20, text="Start Monitoring", font=("Arial", 12), justify="center", bg="white", command=self.start_monitoring)
        self.start_button.pack()
        self.stop_button = tk.Button(self.frame, width=20, text="Stop Monitoring", font=("Arial", 12), justify="center", bg="white", command=self.stop_monitoring)
        self.stop_button.pack()
```

```
#---------------------------------------------------------------------------------------------------------------------------------------------

class System:
    def __init__(self):
        self.window = tk.Tk()
        self.window.title("System status display")

        self.frame = tk.Frame(self.window, width=1200, height=700, bg="Black")
        self.frame.pack()

        menubar = Menu(self.window)
        self.window.config(menu=menubar)

        options_menu = Menu(menubar, tearoff=0)
        menubar.add_cascade(label="Options", menu=options_menu)

        self.display_types = {
            "Overall": OverallDisplay(self.window, self.frame, plt.subplots()),
            "CPU": CPUDisplay(self.window, self.frame),
            "Memory": MemoryDisplay(self.window, self.frame, plt.subplots()),
            "Network": NetworkDisplay(self.window, self.frame, plt.subplots()),
            "Disk": DiskDisplay(self.window, self.frame, plt.subplots()),
            "Process": ProcessDisplay(self.window, self.frame),
            "Temperature" : TemperatureDisplay(self.window, self.frame, plt.subplot()),
            "Battery" : BatteryDisplay(self.window, self.frame, plt.subplot()),
            "Notification" : Notification(self.window, self.frame)
        }

        for label, display in self.display_types.items():
            options_menu.add_command(label=label, command=display.display)

        self.display_overall()
        self.window.mainloop()

    def display_overall(self):
        self.display_types["Overall"].display()

System()
```