```
from __future__ import annotations
import hf_compat  # MUST be the very first import before any transformers
/ sentence_transformers / torch import

#!/usr/bin/env python3
# Keep the names exactly as you had them. Minimal, necessary fix only:
# - ensure the correct datetime class is imported so
`datetime(y,m,d,...)` works
# - preserve your function name, signature and debug keys exactly
# - leave all other logic unchanged except tiny, safe clarity fixes

from datetime import datetime   # <<-- this is the important fix (was
causing "'module' object is not callable")
import re
from typing import Optional, Tuple, Dict, Any

import sys
import os
import cv2
import tkinter as tk
from tkinter import filedialog
from tkinter.scrolledtext import ScrolledText
import time
import datetime
import queue
import keyboard
import re
import subprocess

sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__),
"modules")))

from assistant import AlfredAssistant
from speech import speech
from AI_assistant import AI_Assistant
from vision import vision
from arduino_com import arduino
from AI_vision import AI_VisionModule
from memory import memory
from Repeat_Last import repeat
from face_tracking import face_tracking
from listen import listen
from home_automation import home_auto
import Alfred_config

from GUI import gui as DesktopGUI
from playsound import playsound

import threading
import multiprocessing
from multiprocessing import Process, Value, Array, Lock
import queue
import random
import ast
```

```
   #------- Standby System -------

import assistant_standby_wait as alfred

STANDBY_TIMEOUT_SECONDS = 60  # 5 minutes

_last_activity_time = time.monotonic()
_standby_triggered = False
_standby_running = False

StandBy_Timer = 0
Stanby_Set_time = 180 # Standby in 3 minutes Adjust to your liking

# Wake words
WAKE_WORDS_STANDBY = ["wake up" ,"hallo","hello, alfred","hey alfred","hi
alfred","can you","what is your", "alfred","yes","yeah","yep","sure"]

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

# Global vars and queues
My_new_GPT_Prompt = 0
Alfred_No_Ollama = 0
Silence_Counter_Quiet = 0
Silence_Counter2 = 0
AddPage = 0
PreviousPage = 1
Memory = []
Do_History = []
Did_History = []
ToDoList = []
MyToDoListEditedWithout = ""
MyToDoListEdited = ""
AlfredQueryOfflineNew = ""
AlfredQueryOfflineToDoList = ""
Alfred_Repeat_Previous_Response = ""
Camera_Input_Channel = 1
stop_flag = False
muted = False  # Global flag for text-to-speech
log_queue = queue.Queue()
##query_queue = queue.Queue()
text = ""
speaker_speak = ""
and_then_speaker = ""


# --- Persistent multi-task queue globals ---
query_queue_then = []   # list of dicts: [{"message":..., "username":...,
"score":None}, ...]
task_index = 0          # next item to process
processing_task = False # are we in the middle of a queued run?
AlfredQueryOffline = "" # must always exist as a string
print("main")
```

```
# --- reminders import (robust) ---

import hashlib


_REMINDERS_AVAILABLE = False
reminders = None

try:
    import scheduled_commands as scheduled_commands
except Exception:
    scheduled_commands = None

  # make sure these globals exist once at module top (declare these once
in the file)
try:
    _last_scheduled_hash
    _last_scheduled_time
except NameError:
    _last_scheduled_hash = None
    _last_scheduled_time = 0.0


# ===== Robust imports for reminders + scheduled_commands  =====
import importlib, importlib.util, sys, os, traceback, glob

def _try_import_names(names):
    for nm in names:
        try:
            m = importlib.import_module(nm)
            print(f"[INFO] Imported module '{nm}'")
            return m, nm
        except Exception as e:
            # don't spam stacktrace for every try; print compact info
            print(f"[DEBUG] import {nm} failed: {e}")
    return None, None

def _try_import_by_path_search(patterns=("reminders*.py",)):
    # look in cwd and a few parent dirs
    cwd = os.getcwd()
    candidates = []
    for root, dirs, files in os.walk(cwd):
        for pat in patterns:
            for f in files:
                if f.lower().startswith("reminders") and
f.lower().endswith(".py"):
                    candidates.append(os.path.join(root, f))
        # small optimisation: don't walk too deep
        if len(candidates) > 10:
            break
    if not candidates:
```

```
        # also check project root (one level up)
        parent = os.path.dirname(cwd)
        for f in glob.glob(os.path.join(parent, "reminders*.py")):
            candidates.append(f)
    if not candidates:
        return None
    # prefer files in cwd
    candidates.sort(key=lambda p: (0 if os.path.dirname(p) == cwd else 1,
p))
    for path in candidates:
        try:
            name = "reminders_from_path"
            spec = importlib.util.spec_from_file_location(name, path)
            mod = importlib.util.module_from_spec(spec)
            spec.loader.exec_module(mod)
            print(f"[INFO] Imported reminders from path: {path}")
            return mod
        except Exception as e:
            print(f"[WARN] Failed to import reminders from {path}: {e}")
            traceback.print_exc()
    return None

# Try common names first
reminders = None
scheduled_commands = None

reminder_names = ["reminders", "reminders_module", "modules.reminders",
"modules.reminders_module"]
reminders, used_name = _try_import_names(reminder_names)
if reminders is None:
    # fallback: try to find a file named reminders*.py and import it
    reminders = _try_import_by_path_search(("reminders*.py",))
    if reminders is None:
        print("[WARN] Could not import reminders by name or file search.
Make sure the file 'reminders_module.py' or 'reminders.py' is in your
project and on PYTHONPATH.")

# scheduled_commands: try common names (you already have this one
working)
sched_names = ["scheduled_commands", "modules.scheduled_commands"]
scheduled_commands, used_name_sched = _try_import_names(sched_names)
if scheduled_commands is None:
    # attempt file search similar to reminders
    # look for scheduled_commands*.py
    cwd = os.getcwd()
    found = None
    for root, dirs, files in os.walk(cwd):
        for f in files:
            if f.lower().startswith("scheduled_commands") and
f.lower().endswith(".py"):
                found = os.path.join(root, f)
                break
        if found: break
    if found:
```

```
        try:
            spec =
importlib.util.spec_from_file_location("scheduled_commands_from_path",
found)
            scheduled_commands = importlib.util.module_from_spec(spec)
            spec.loader.exec_module(scheduled_commands)
            print(f"[INFO] Imported scheduled_commands from path:
{found}")
        except Exception as e:
            scheduled_commands = None
            print(f"[WARN] Failed to import scheduled_commands from
{found}: {e}")
            traceback.print_exc()
    else:
        print("[WARN] Could not import scheduled_commands by name or file
search. Ensure scheduled_commands.py exists in your project.")

# Start threads if present (safe-guarded)

# near other imports at top of file
from reminders_module import get_reminder_speaking_flag

try:
    if reminders is not None and hasattr(reminders,
"start_reminder_notifier_thread"):
        try:
            reminders.start_reminder_notifier_thread()
            print("[INFO] reminders notifier thread start requested.")
        except Exception as e:
            print("[WARN] reminders.start_reminder_notifier_thread()
error:", e)
            traceback.print_exc()
except Exception:
    pass

try:
    if scheduled_commands is not None and hasattr(scheduled_commands,
"start_scheduler_thread"):
        try:
            scheduled_commands.start_scheduler_thread()
            print("[INFO] scheduled_commands scheduler thread start
requested.")
        except Exception as e:
            print("[WARN] scheduled_commands.start_scheduler_thread()
error:", e)
            traceback.print_exc()
except Exception:
    pass

# OPTIONAL: wrap scheduled_commands.handle_command_text to print debug
info
# Uncomment if you want verbose parsing logs for troubleshooting.
"""
```

```
if scheduled_commands is not None and hasattr(scheduled_commands,
"handle_command_text"):
    _orig_handle = scheduled_commands.handle_command_text
    def _debug_handle(text, gui=None):
        print("[DEBUG scheduled_commands] handle_command_text called
with:", repr(text))
        res = None
        try:
            res = _orig_handle(text, gui=gui)
            print("[DEBUG scheduled_commands] returned:", res)
        except Exception as e:
            print("[DEBUG scheduled_commands] exception:", e)
            traceback.print_exc()
            raise
        return res
    scheduled_commands.handle_command_text = _debug_handle
    print("[DEBUG] scheduled_commands.handle_command_text wrapped for
logging.")
"""
#
================================================================================
============


# ------------------------------------------------------------
# Delegation helpers (REMINDERS & TIMED COMMANDS)
# ------------------------------------------------------------

_last_scheduled_hash = None
_last_scheduled_time = 0.0




def _strip_main_timestamp_wrapper(s: str) -> str:
    if not s:
        return s
    try:
        s2 = re.sub(r"^\s*\d{4}-\d{2}-
\d{2}\s*:\s*\d{2}:\d{2}:\d{2}\s*:\s*", "", str(s))
        s2 = re.sub(r"\s*:\s*[A-Za-z0-9\-'
]{1,40}\s*(Home|Office|Work|Phone)?\s*$", "", s2, flags=re.I)
        return s2.strip()
    except Exception:
        return str(s).strip()

def handle_reminder_delegation(AlfredQueryOffline, gui) -> bool:
    """
    Returns True if the utterance was handled as a reminder (so caller
should continue loop).
    """
    try:
        if not AlfredQueryOffline:
            return False
```

```
        lower_text = str(AlfredQueryOffline).lower()

        reminder_triggers = (
            "remind me", "create a reminder", "create me a reminder",
            "set a reminder", "set reminder", "i want to remember",
            "can you remember", "what can you remember", "what do i
have",
            "what are my reminders", "what is my schedule", "what do i
have scheduled",
            "what did i tell you", "what are my meetings", "schedule me a
meeting with",
            "create me a meeting", "setup a meeting", "create me a
reminder"
        )

        if not any(t in lower_text for t in reminder_triggers):
            return False  # not a reminder

        cleaned_msg = _strip_main_timestamp_wrapper(AlfredQueryOffline)

        if reminders is not None and hasattr(reminders,
"handle_voice_command"):
            _old_suppress = None
            try:
                _old_suppress = getattr(speech, "_suppress_auto_listen",
None)
            except Exception:
                _old_suppress = None
            try:
                try:
                    setattr(speech, "_suppress_auto_listen", True)
                except Exception:
                    pass
                try:
                    reminders.handle_voice_command(cleaned_msg, gui=gui)
                except Exception as e:
                    print("Reminder handler runtime error:", e)
                    traceback.print_exc()
                    try:
                        if speech is not None and hasattr(speech,
"AlfredSpeak"):
                            speech.AlfredSpeak("Sorry, I could not create
the reminder due to an internal error.")
                    except Exception:
                        pass
            finally:
                try:
                    if _old_suppress is None:
                        if hasattr(speech, "_suppress_auto_listen"):
                            try: delattr(speech, "_suppress_auto_listen")
                            except Exception: pass
                    else:
                        try: setattr(speech, "_suppress_auto_listen",
_old_suppress)
```

```
                        except Exception: pass
                except Exception:
                    pass
        else:
            print("Reminder requested but reminders module not
available.")
            try:
                if speech is not None and hasattr(speech, "AlfredSpeak"):
                    speech.AlfredSpeak("Sorry, the reminder feature is
not available right now.")
            except Exception:
                pass

        # mark handled
        try:
            # consume the utterance in caller scope by signaling True
            if MyToDoListEdited != []:
                MyToDoListEdited.pop(0)
                AlfredQueryOfflineNew = MyToDoListEdited
        except Exception:
            pass
        return True

    except Exception as e:
        print("handle_reminder_delegation error:", e)
        traceback.print_exc()
        return False

import re

import re

def strip_leading_timestamp_line(s: str) -> str:
    """
    Remove a leading tag and timestamp from a line like:
      "New_Message Timed Command       : 2026-01-22 : 19:27:27 : can you
switch the light on please in 5 minutes."
    Returns the message part only:
      "can you switch the light on please in 5 minutes."

    Improvements:
      - Handles variants where date and time appear anywhere in the
string.
      - Ensures there is an explicit separator colon after the time
before taking the remainder as the message.
      - Avoids leaving partial time tokens like "11:" at the start of the
message.
    """
    if not s:
        return ""

    s = s.strip()

    # 1) Exact pattern: prefix : YYYY-MM-DD : HH:MM:SS : message
```

```
    m = re.match(r'^[^:]*:\s*(\d{4}-\d{2}-
\d{2})\s*:\s*(\d{1,2}:\d{2}:\d{2})\s*:\s*(.*)$', s)
    if m:
        return m.group(3).strip()

    # 2) Date+time together then colon: prefix : "YYYY-MM-DD HH:MM:SS" :
message
    m2 = re.match(r'^[^:]*:\s*(\d{4}-\d{2}-
\d{2}\s+\d{1,2}:\d{2}:\d{2})\s*:\s*(.*)$', s)
    if m2:
        return m2.group(2).strip()

    # 3) Time anywhere followed by colon (most robust): " ... 19:27:27 :
message"
    m3 = re.search(r'(\d{1,2}:\d{2}:\d{2})\s*:\s*(.*)$', s)
    if m3:
        return m3.group(2).strip()

    # 4) Date anywhere followed by colon then message (no explicit time)
    m4 = re.search(r'(\d{4}-\d{2}-\d{2})\s*:\s*(.*)$', s)
    if m4:
        # if remaining starts with a time-like token with no separating
colon, try to remove it too
        rem = m4.group(2).strip()
        m_rem_time = re.match(r'^\s*(\d{1,2}:\d{2}:\d{2})\s*(.*)$', rem)
        if m_rem_time:
            # if there's no colon after the time, the time might be glued
to message, so take group(2)
            return m_rem_time.group(2).strip()
        return rem

    # 5) Fallback heuristic: split on " : " and return the last segment,
    #    but only if that last segment looks like normal text (contains
letters).
    parts = [p.strip() for p in re.split(r'\s*:\s*', s) if p is not None]
    if len(parts) >= 2:
        candidate = parts[-1]
        # ensure candidate isn't something like just digits (avoid
returning leftover time digits)
        if re.search(r'[A-Za-z0-9]', candidate) and not
re.fullmatch(r'\d{1,2}', candidate):
            return candidate

    # Final fallback: return original trimmed string
    return s



def handle_timed_command_delegation(AlfredQueryOffline, gui,
debounce_window: float = 5.0) -> bool:
    """
    Returns True if the utterance was handled as a timed command (so
caller should continue loop).
```

```
    If scheduling fails, it will speak an error message and still return
True to block immediate execution.
    This version asks the user to confirm the cleaned command before
scheduling.
    """
    try:
        if not AlfredQueryOffline:
            return False

        # time detection: covers many variants (7 o clock, 7:00, half
past six, tomorrow, in 5 minutes, etc.)
        time_pattern = re.compile(
            r"\b(?:at\b|o'clock\b|o clock\b|oclock\b|half past\b|quarter
past\b|quarter to\b|"

r"in\s+\d+\s+(?:minutes|minute|hours|hour|seconds|second)\b|tomorrow\b|to
day\b|noon\b|midnight\b|"
            r"\d{1,2}:\d{2}\b|\d{1,2}\s*(?:o'clock|o
clock|oclock)\b|(?:one|two|three|four|five|six|seven|eight|nine|ten|eleve
n|twelve)\s*(?:o'clock|o clock|oclock)\b)\b",
            flags=re.I
        )
        if not time_pattern.search(str(AlfredQueryOffline)):
            return False  # not a timed command

        # parse/normalize input into message + optional metadata
        message, speaker, score, gender, gender_conf, timestamp =
extract_text_from_timed_command(AlfredQueryOffline)

        New_Message = message
        New_speaker = speaker
        New_score = score
        New_gender = gender
        New_gender_conf = gender_conf
        New_timestamp = timestamp

        print(f"New_Message Timed Command       : {New_Message}")
        print(f"New_speaker Timed Command       : {New_speaker}")
        print(f"New_score Timed Command         : {New_score}")
        print(f"New_gender Timed Command        : {New_gender}")
        print(f"New_gender_conf Timed Command   : {New_gender_conf}")
        print(f"New_timestamp Timed Command     : {New_timestamp}")

        New_Message_No_Timestamp =
strip_leading_timestamp_line(New_Message)
        print(f"New_Message_No_Timestamp Timed Command     :
{New_Message_No_Timestamp}")

        cleaned_msg =
_strip_main_timestamp_wrapper(New_Message_No_Timestamp)

        # ----------------------- confirmation helper ------------------
------
```

```
        def _ask_confirm_and_listen_local(prompt_text: str, attempts: int
= 3, timeout_per_attempt: int = 12) -> bool:
            """
            Ask the user to confirm prompt_text. Returns True for
confirmed, False for not confirmed.
            Accepts clear yes/no responses such as:
              - YES: "yes that is correct", "yes", "yeah", "yep", "sure",
"correct"
              - NO : "that is not correct", "that is incorrect", "no",
"nope", "not correct", "incorrect"
            Uses speech.AlfredSpeak() to ask and listen.listen() to
receive responses.
            Defensive: coerce responses to string before .lower().
            """
            yes_tokens = {"yes, that is correct", "yes thank you", "yes
you are correct", "yes that is great",
                          "yes", "yeah", "yep", "sure", "correct",
"affirmative"}
            no_tokens = {"no, that is not correct", "that is not
correct", "no, that is incorrect", "no", "nope", "not correct",
"incorrect", "don't", "do not", "dont"}

            # speak the confirmation prompt
            ask_text = f"Confirm: {prompt_text}. Say 'yes that is
correct' or say 'that is not correct'."
            try:
                if speech is not None and hasattr(speech, "AlfredSpeak"):
                    speech.AlfredSpeak(ask_text)
                else:
                    print("[confirm ask]", ask_text)
            except Exception:
                print("[confirm ask fail]")

            # Try to get a response attempts times
            for attempt in range(attempts):
                try:
                    resp = ""
                    # first try the project's listen() if available
                    if listen is not None and hasattr(listen, "listen"):
                        try:
                            raw = listen.listen()
                        except Exception:
                            raw = ""
                        # coerce safely
                        if raw is None:
                            resp = ""
                        else:
                            resp = str(raw).strip()
                    else:
                        # fallback to input() for debugging environments
                        try:
                            raw = input("(confirm) say 'yes' or 'no': ")
                        except Exception:
                            raw = ""
```

```
                        resp = "" if raw is None else str(raw).strip()

                    if not resp:
                        # re-prompt once or wait briefly
                        if attempt + 1 < attempts:
                            try:
                                if speech is not None and hasattr(speech,
"AlfredSpeak"):
                                    speech.AlfredSpeak("I didn't catch
that. Please say yes that is correct, or say that is not correct.")
                                else:
                                    print("(confirm) I didn't catch that.
Please say yes or no.")
                            except Exception:
                                pass
                            # small pause before next attempt (optional)
                            try:
                                time.sleep(0.6)
                            except Exception:
                                pass
                            continue

                    low = resp.lower().strip()

                    # exact phrase matches first
                    if any(tok in low for tok in yes_tokens) and not
any(tok in low for tok in no_tokens):
                        return True
                    if any(tok in low for tok in no_tokens) and not
any(tok in low for tok in yes_tokens):
                        return False

                    # fallback: check first token
                    tokens = re.findall(r"[a-z]+", low)
                    if tokens:
                        if tokens[0] in yes_tokens:
                            return True
                        if tokens[0] in no_tokens:
                            return False

                    # not understood -> reprompt
                    if attempt + 1 < attempts:
                        try:
                            if speech is not None and hasattr(speech,
"AlfredSpeak"):
                                speech.AlfredSpeak("Please answer with
'yes that is correct' or 'that is not correct'.")
                            else:
                                print("(confirm) please answer 'yes' or
'no'.")
                        except Exception:
                            pass
                        try:
                            time.sleep(0.6)
```

```
                        except Exception:
                            pass
                        continue

                except Exception as e:
                    print("_ask_confirm_and_listen_local error:", e)
                    traceback.print_exc()
                    continue

            # default to not confirmed if we didn't parse a clear
response
            return False

        # ---------------------- end confirmation helper ----------------
------

        # Ask the user to confirm the cleaned command before scheduling
        try:
            confirmed = _ask_confirm_and_listen_local(cleaned_msg,
attempts=3, timeout_per_attempt=12)
        except Exception as e:
            print("Confirmation prompt failed:", e)
            traceback.print_exc()
            confirmed = False

        if not confirmed:
            # user said it's incorrect (or we couldn't parse a clear
'yes'); cancel scheduling
            try:
                if speech is not None and hasattr(speech, "AlfredSpeak"):
                    speech.AlfredSpeak("Okay — I will not schedule that.
If you'd like, please say the command again with the time.")
                else:
                    print("[scheduled_commands] User did not confirm;
scheduling canceled.")
            except Exception:
                pass

            # consume the utterance (prevents immediate execution)
            try:
                if MyToDoListEdited != []:
                    MyToDoListEdited.pop(0)
                    AlfredQueryOfflineNew = MyToDoListEdited
            except Exception:
                pass
            return True

        # If confirmed, proceed with debounce + scheduling (existing
behavior)

        # debounce globals (module-level)
        global _last_scheduled_hash, _last_scheduled_time
        try:
```

```
            h = hashlib.md5((cleaned_msg or "").encode("utf-
8")).hexdigest()
            now_ts = time.time()
            if _last_scheduled_hash == h and (now_ts -
_last_scheduled_time) < float(debounce_window):
                print("[scheduled_commands] duplicate scheduling avoided
(debounced).")
                try:
                    if MyToDoListEdited != []:
                        MyToDoListEdited.pop(0)
                        AlfredQueryOfflineNew = MyToDoListEdited
                except Exception:
                    pass
                return True  # consumed (debounced)
        except Exception as e:
            print("Debounce guard error (unexpected):", e)
            traceback.print_exc()

        # delegate to scheduled_commands
        scheduled_returned = None
        if scheduled_commands is not None and hasattr(scheduled_commands,
"handle_command_text"):
            _old_suppress = None
            try:
                _old_suppress = getattr(speech, "_suppress_auto_listen",
None)
            except Exception:
                _old_suppress = None
            try:
                try:
                    setattr(speech, "_suppress_auto_listen", True)
                except Exception:
                    pass
                try:
                    scheduled_returned =
scheduled_commands.handle_command_text(cleaned_msg, gui=gui)
                except Exception as e:
                    print("scheduled_commands.handle_command_text runtime
error:", e)
                    traceback.print_exc()
                    scheduled_returned = None
            finally:
                try:
                    if _old_suppress is None:
                        if hasattr(speech, "_suppress_auto_listen"):
                            try: delattr(speech, "_suppress_auto_listen")
                            except Exception: pass
                    else:
                        try: setattr(speech, "_suppress_auto_listen",
_old_suppress)
                        except Exception: pass
                except Exception:
                    pass
        else:
```

```
            print("Time-based command requested but scheduled_commands
module not available.")
            try:
                if speech is not None and hasattr(speech, "AlfredSpeak"):
                    speech.AlfredSpeak("Sorry, the scheduling feature is
not available right now.")
            except Exception:
                pass
            scheduled_returned = None

        if scheduled_returned:
            # stamp debounce
            try:
                _last_scheduled_hash = hashlib.md5((cleaned_msg or
"").encode("utf-8")).hexdigest()
                _last_scheduled_time = time.time()
            except Exception:
                pass

            try:
                if MyToDoListEdited != []:
                    MyToDoListEdited.pop(0)
                    AlfredQueryOfflineNew = MyToDoListEdited
            except Exception:
                pass
            # scheduling succeeded -> consumed
            return True

        # scheduling failed: inform user and still block immediate
execution (safer)
        try:
            msg = "I couldn't schedule that. Please say 'switch the light
on at seven o'clock' or 'switch the light on in five minutes'."
            if speech is not None and hasattr(speech, "AlfredSpeak"):
                speech.AlfredSpeak(msg)
            else:
                print("[scheduled_commands] " + msg)
        except Exception:
            pass

        try:
            if MyToDoListEdited != []:
                MyToDoListEdited.pop(0)
                AlfredQueryOfflineNew = MyToDoListEdited
        except Exception:
            pass

        # block immediate execution (user must reissue if they want
immediate action)
        return True

    except Exception as e:
        print("handle_timed_command_delegation error:", e)
        traceback.print_exc()
```

```
        # don't block main loop on unexpected error — allow normal flow
        return False

# global handle (if you used the subprocess approach anywhere)
OLLAMA_PROCESS = globals().get("OLLAMA_PROCESS", None)

import sys
from modules.GUI import gui as DesktopGUI
import webui
from shared_queues import query_queue, log_queue

from face_tracking import Vision_Who_InFront_Look_At_New

from shutdown_helpers import app_shutdown_event
import shutdown_helpers as sh
import shutil

# Global variable for current speaker
New_speaker = None
_speaker_lock = threading.Lock()

def set_current_speaker(name: str):
    global New_speaker
    with _speaker_lock:
        New_speaker = name

def get_current_speaker():
    with _speaker_lock:
        return New_speaker

START_TIME_KEYWORDS = [
    "what is",
    "tell me"
    ]

TIME_KEYWORDS = [
    "the time"
]

START_DATE_KEYWORDS = [
    "what is",
    "tell me"
    ]

DATE_KEYWORDS = [
    "the date today",
    "the current date"
]

START_DAY_KEYWORDS = [
    "what is",
    "tell me"
    ]
```

```
DAY_KEYWORDS = [
    "the day"
]

# Words/phrases that should trigger pause
_pause_triggers = [
    "silence", "wait", "shut up", "hold on", "keep quiet", "shush",
"please pause", "pause please", "pause"
]

import re
import base64

def extract_text_from_timed_command(query):
    """
    Returns: (message, speaker, score, gender, gender_conf, timestamp)

    Kept your original behavior; only added safer parsing after a
detected
    timestamp to avoid splitting on every ':' and causing list-index
errors.
    """
    if query is None:
        return "", None, None, None, None, None

    import re, base64

    def _extract_timestamp(fragment: str):
        if not fragment:
            return None
        m_date = re.search(r'(?P<date>\d{4}-\d{2}-\d{2})', fragment)
        m_time = re.search(r'(?P<time>\d{2}:\d{2}:\d{2})', fragment)
        if m_date and m_time:
            return f"{m_date.group('date')} {m_time.group('time')}"
        if m_date:
            return m_date.group('date')
        m_date2 = re.search(r'(?P<date>\d{2}/\d{2}/\d{4})', fragment)
        if m_date2:
            return m_date2.group('date')
        return None

    def _norm(v):
        if v is None:
            return None
        vs = str(v).strip()
        if vs.lower() in ("none", "null", ""):
            return None
        return vs

    # ---------- dict case ----------
    if isinstance(query, dict):
        text_ = query.get("text") or query.get("query") or
query.get("message") or query.get("q") or ""
        speaker_ = query.get("username") or query.get("speaker")
```

```
        score_ = query.get("score")
        gender_ = query.get("gender")
        gender_conf_ = query.get("gender_conf")
        timestamp_ = query.get("timestamp") or query.get("time") or
query.get("date") or None
        return (
            str(text_).strip(),
            (str(speaker_).strip() if speaker_ is not None else None),
            _norm(score_),
            _norm(gender_),
            _norm(gender_conf_),
            (str(timestamp_).strip() if timestamp_ is not None else None)
        )

    # --- string case ---
    if isinstance(query, str):
        s = query.strip()

        # base64 decode heuristic
        try:
            if len(s) > 50 and re.fullmatch(r'[A-Za-z0-9+/=\s]+', s) and
'\n' not in s:
                try:
                    decoded = base64.b64decode(s).decode('utf-8')
                    if decoded:
                        s = decoded.strip()
                except Exception:
                    pass
        except Exception:
            pass

        s = s.strip()

        # Helper to extract metadata by regex from a fragment
        def _extract_kv_from_fragment(fragment: str):
            """Return normalized values for score, gender, gender_conf,
speaker (if found)."""
            score = gender = gender_conf = speaker = None
            if not fragment:
                return speaker, score, gender, gender_conf
            # find kv pairs using regex
            m_score =
re.search(r"'score'\s*[:=]\s*(['\"]?)(?P<v>[^'\"\s:]+)\1", fragment,
flags=re.IGNORECASE)
            if m_score:
                score = _norm(m_score.group('v'))
            m_gender =
re.search(r"'gender'\s*[:=]\s*(['\"]?)(?P<v>[^'\"\s:]+)\1", fragment,
flags=re.IGNORECASE)
            if m_gender:
                gender = _norm(m_gender.group('v'))
            m_gender_conf =
re.search(r"'gender_conf'\s*[:=]\s*(['\"]?)(?P<v>[^'\"\s:]+)\1",
fragment, flags=re.IGNORECASE)
```

```
            if m_gender_conf:
                gender_conf = _norm(m_gender_conf.group('v'))

            # Find a trailing username that doesn't look like a kv pair
            # Strategy: look for last colon-separated segment that does
NOT contain "score" or "gender"
            # or look for a plain token at the end.
            m_last = re.search(r'[:\s]\s*([^:]+)\s*$', fragment)
            if m_last:
                candidate = m_last.group(1).strip(" '\"")
                if candidate and not
re.search(r"\b(score|gender|gender_conf)\b", candidate,
flags=re.IGNORECASE):
                    # be conservative: if candidate looks like a simple
token (no spaces or short) accept it
                    if " " not in candidate or len(candidate) <= 32:
                        speaker = candidate

            # Also accept explicit "'username': Name" pattern
            m_user =
re.search(r"'username'\s*[:=]\s*['\"]?(?P<u>[^'\"\n:]+)['\"]?", fragment,
flags=re.IGNORECASE)
            if m_user:
                speaker = m_user.group('u').strip()

            return speaker, score, gender, gender_conf

        # detect triple fences first (unchanged)
        first_triple = s.find("'''")
        backtick_first = s.find("```")
        wrapper_pos = None
        wrapper_token = None
        if first_triple != -1:
            wrapper_pos = first_triple; wrapper_token = "'''"
        if backtick_first != -1 and (wrapper_pos is None or
backtick_first < wrapper_pos):
            wrapper_pos = backtick_first; wrapper_token = "```"

        if wrapper_pos is not None:
            start = wrapper_pos
            end = s.find(wrapper_token, start + len(wrapper_token))
            if end != -1:
                inner = s[start + len(wrapper_token): end]
                remainder = s[end + len(wrapper_token):].strip()
                inner = inner.lstrip()
                inner = re.sub(r"^'+", "", inner)
                inner =
re.sub(r"^\s*(?:'message'\s*:\s*|\"message\"\s*:\s*|message\s*:)\s*", "",
inner, flags=re.IGNORECASE)
                speaker, score, gender, gender_conf =
_extract_kv_from_fragment(remainder)
                timestamp = _extract_timestamp(remainder) or
_extract_timestamp(s)
                if not speaker:
```

```
                    m_all =
re.search(r"'username'\s*[:=]\s*['\"]?(?P<u>[^'\"\n:]+)['\"]?",
s[end+len(wrapper_token):])
                    if m_all:
                        speaker = m_all.group("u").strip()
                message = inner.strip()
                return message, (speaker if speaker else None), (score if
score else None), (gender if gender else None), (gender_conf if
gender_conf else None), (timestamp if timestamp else None)

        # ---------------- SAFER TIMESTAMP-LEAD PARSER ----------------
        # Find date/time anywhere (YYYY-MM-DD : HH:MM:SS)
        dt_match = re.search(r'(?P<date>\d{4}-\d{2}-
\d{2})\s*:\s*(?P<time>\d{2}:\d{2}:\d{2})', s)
        if dt_match:
            date = dt_match.group('date')
            time_ = dt_match.group('time')
            timestamp = f"{date} {time_}"

            # remainder after the date/time match
            rem = s[dt_match.end():].strip()

            # If there is an explicit metadata key
(score/gender/gender_conf) somewhere in rem,
            # split message = rem up to that key, and parse the remainder
as metadata.
            meta_key_match =
re.search(r"(?:'score'|'gender_conf'|'gender'|\"score\"|\"gender_conf\"|\
"gender\")\s*[:=]", rem, flags=re.IGNORECASE)
            if meta_key_match:
                message_candidate = rem[:meta_key_match.start()].strip("
:\n\t")
                meta_fragment = rem[meta_key_match.start():].strip()
            else:
                # fallback: split on conservative delimiter " : " (space-
colon-space)
                parts = [p.strip() for p in re.split(r'\s+:\s+', rem) if
p.strip()]
                if parts:
                    message_candidate = parts[0]
                    meta_fragment = ' : '.join(parts[1:]) if len(parts) >
1 else ""
                else:
                    message_candidate = rem
                    meta_fragment = ""

            # Clean message if it ends with common metadata tokens
accidentally included
            message_candidate = re.sub(r"(\s*'?score'?\s*[:=]\s*[^:]+)$",
"", message_candidate, flags=re.IGNORECASE).strip()
            message_candidate =
re.sub(r"(\s*'?gender'?\s*[:=]\s*[^:]+)$", "", message_candidate,
flags=re.IGNORECASE).strip()
```

```
            message_candidate =
re.sub(r"(\s*'?gender_conf'?\s*[:=]\s*[^:]+)$", "", message_candidate,
flags=re.IGNORECASE).strip()

            # extract kvs from meta_fragment safely
            speaker, score, gender, gender_conf =
_extract_kv_from_fragment(meta_fragment)

            # If speaker not found in kvs, check if the very last token
in rem is a plain username
            if not speaker and rem:
                # try last colon-separated token (conservative)
                last_token_match =
re.search(r'(?:[:\s]+)(?P<t>[^:]+)\s*$', rem)
                if last_token_match:
                    last_token = last_token_match.group('t').strip("
'\"")
                    if last_token and not
re.search(r"\b(score|gender|gender_conf)\b", last_token,
flags=re.IGNORECASE):
                        # ensure last_token isn't part of a key:value by
checking colon presence earlier
                        # accept simple username-style tokens
                        if " " not in last_token or len(last_token) <=
32:
                            speaker = last_token

            return message_candidate.strip(), (speaker if speaker else
None), (score if score else None), (gender if gender else None),
(gender_conf if gender_conf else None), timestamp

        # ---------- older structured pattern ----------
        m_struct = re.match(
            r'^(?P<prefix>[^:]+?)\s*:\s*(?P<date>\d{4}-\d{2}-
\d{2})\s*:\s*(?P<time>\d{2}:\d{2}:\d{2})\s*:\s*(?P<msg>.*?)\s*:\s*(?P<use
r>.+)$',
            s
        )
        if m_struct:
            date = m_struct.group('date'); time_ = m_struct.group('time')
            timestamp = f"{date} {time_}"
            message = m_struct.group('msg').strip()
            user_full = m_struct.group('user').strip()
            gender = None
            m_gender = re.search(r'\((?P<g>[^)]+)\)\s*$', user_full)
            if m_gender:
                gender = m_gender.group('g').strip()
                user_clean = re.sub(r'\s*\([^)]+\)\s*$', '',
user_full).strip()
            else:
                user_clean = user_full
            speaker = user_clean if user_clean else None
            return message, speaker, None, (gender if gender else None),
None, timestamp
```

```
        # structured string attempt (unchanged)
        try:
            pattern =
r"'message'\s*:\s*'(?P<message>.*?)'\s*:\s*'username'\s*:\s*'(?P<username
>[^']*)'(?:\s*:\s*'score'\s*:\s*'(?P<score>[^']*)')?(?:\s*:\s*'gender'\s*
:\s*'(?P<gender>[^']*)')?(?:\s*:\s*'gender_conf'\s*:\s*'(?P<gender_conf>[
^']*)')?"
            m = re.search(pattern, s, flags=re.DOTALL)
            if m:
                message_ = m.group('message').strip()
                speaker_ = m.group('username').strip() or None
                score_ = m.group('score') or None
                gender_ = m.group('gender') or None
                gender_conf_ = m.group('gender_conf') or None
                timestamp_ = _extract_timestamp(s)
                return message_, speaker_, _norm(score_), _norm(gender_),
_norm(gender_conf_), timestamp_
        except Exception:
            pass

        # username anywhere (unchanged)
        m_user_any =
re.search(r"'username'\s*[:=]\s*['\"]?(?P<u>[^'\"\n:]+)['\"]?", s)
        speaker = None; score = None; gender = None; gender_conf = None
        timestamp = _extract_timestamp(s)
        if m_user_any:
            speaker = m_user_any.group("u").strip()
            s = (s[: m_user_any.start()] + s[m_user_any.end():]).strip("
:\n\t ")
            ms =
re.search(r"'score'\s*[:=]\s*['\"]?(?P<s>[^'\"\s:]+)['\"]?", s,
flags=re.IGNORECASE)
            if ms:
                score = _norm(ms.group("s").strip())
            mg =
re.search(r"'gender'\s*[:=]\s*['\"]?(?P<g>[^'\"\s:]+)['\"]?", s,
flags=re.IGNORECASE)
            if mg:
                gender = _norm(mg.group("g").strip())
            mgc =
re.search(r"'gender_conf'\s*[:=]\s*['\"]?(?P<gc>[^'\"\s:]+)['\"]?", s,
flags=re.IGNORECASE)
            if mgc:
                gender_conf = _norm(mgc.group("gc").strip())
            candidate = s.strip()
            return candidate, (speaker if speaker else None), (score if
score else None), (gender if gender else None), (gender_conf if
gender_conf else None), (timestamp if timestamp else None)

        # username at end (unchanged)
        m_user2 =
re.search(r"(?:\busername\b|\buser\b)\s*[:=]\s*['\"]?(?P<u>[^'\"\n]+)['\"
]?\s*$", s, flags=re.IGNORECASE)
```

```
        timestamp = _extract_timestamp(s)
        if m_user2:
            speaker = m_user2.group("u").strip()
            message_candidate =
re.sub(r"(?:[:\s]*\busername\b\s*[:=]\s*['\"]?.+?['\"]?\s*$)|(?:[:\s]*\bu
ser\b\s*[:=]\s*['\"]?.+?['\"]?\s*$)", "", s, flags=re.IGNORECASE).strip("
:")
            return message_candidate, (speaker if speaker else None),
None, None, None, (timestamp if timestamp else None)

        # last-token username heuristic (unchanged)
        m_user3 =
re.match(r"^(?P<body>.*\S)\s*:\s*(?P<u>[^\n:]{1,48})\s*$", s)
        timestamp = _extract_timestamp(s)
        if m_user3:
            maybe_u = m_user3.group("u").strip()
            maybe_body = m_user3.group("body").strip()
            if " " not in maybe_u or len(maybe_u) <= 24:
                return maybe_body, maybe_u, None, None, None, (timestamp
if timestamp else None)

        # fallback
        return s, None, None, None, None, None

    # fallback for non-strings
    return str(query).strip(), None, None, None, None, None


def extract_text_from_query(query):
    """
    Returns: (message, speaker, score, gender, gender_conf)

    Behavior:
      - dict: uses keys text/query/message/q and username/speaker
      - strings:
        * will detect and extract triple-single-quote wrappers '''...'''
(or ``` fences)
        * if wrapper present: returns inner content (cleaned) as message
and parses remainder for username/score/gender/gender_conf
        * if no wrapper: tries safe structured-string regex for
'message':'...':'username':'name' patterns
        * falls back to returning raw string (and any username if
parseable)
      - tries base64 decode if the input looks like base64
    """
    if query is None:
        return "", None, None, None, None

    # --- dict case (preferred structured format) ---
    if isinstance(query, dict):
        text_ = query.get("text") or query.get("query") or
query.get("message") or query.get("q") or ""
        speaker_ = query.get("username") or query.get("speaker")
        score_ = query.get("score")
```

```
        gender_ = query.get("gender")
        gender_conf_ = query.get("gender_conf")

        # normalize to str
        return str(text_).strip(), (str(speaker_).strip() if speaker_ is
not None else None), score_, gender_, gender_conf_

    # --- string case ---
    if isinstance(query, str):
        s = query.strip()

        # 1) Try base64 decode heuristic (optional)
        try:
            if len(s) > 50 and re.fullmatch(r'[A-Za-z0-9+/=\s]+', s) and
'\n' not in s:
                try:
                    decoded = base64.b64decode(s).decode('utf-8')
                    if decoded:
                        s = decoded.strip()
                except Exception:
                    pass
        except Exception:
            pass

        # normalize repeated leading spaces/newlines
        s = s.strip()

        # Helper to extract metadata (username/score/gender/gender_conf)
from a text fragment
        def _extract_meta(fragment):
            frag = fragment or ""
            frag = str(frag)
            speaker = None
            score = None
            gender = None
            gender_conf = None

            # 1) explicit 'username':Name
            m =
re.search(r"'username'\s*:\s*['\"]?(?P<u>[^'\"\n:]+)['\"]?", frag)
            if m:
                speaker = m.group("u").strip()

            # 2) username: Name or user: Name (end-of-string preferred)
            if speaker is None:
                m2 =
re.search(r"(?:\busername\b|\buser\b)\s*[:=]\s*['\"]?(?P<u>[^'\"\n]+)['\"
]?\s*$", frag, flags=re.IGNORECASE)
                if m2:
                    speaker = m2.group("u").strip()

            # 3) last-token " : Name" heuristic (conservative)
            if speaker is None:
```

```
                m3 =
re.match(r"^(?P<body>.*\S)\s*:\s*(?P<u>[^\n:]{1,48})\s*$", frag)
                if m3:
                    maybe_u = m3.group("u").strip()
                    if " " not in maybe_u or len(maybe_u) <= 24:
                        speaker = maybe_u

            # 4) numeric/word tokens for score/gender/gender_conf if
present
            ms =
re.search(r"'score'\s*:\s*['\"]?(?P<s>[^'\"\s:]+)['\"]?", frag,
flags=re.IGNORECASE)
            if ms:
                score = ms.group("s").strip()
            mg =
re.search(r"'gender'\s*:\s*['\"]?(?P<g>[^'\"\s:]+)['\"]?", frag,
flags=re.IGNORECASE)
            if mg:
                gender = mg.group("g").strip()
            mgc =
re.search(r"'gender_conf'\s*:\s*['\"]?(?P<gc>[^'\"\s:]+)['\"]?", frag,
flags=re.IGNORECASE)
            if mgc:
                gender_conf = mgc.group("gc").strip()

            return speaker, score, gender, gender_conf

        # 2) Detect triple-single-quote wrapper '''...'''
        first_triple = s.find("'''")
        backtick_first = s.find("```")
        # choose wrapper if present earliest
        wrapper_pos = None
        wrapper_token = None
        if first_triple != -1:
            wrapper_pos = first_triple
            wrapper_token = "'''"
        if backtick_first != -1 and (wrapper_pos is None or
backtick_first < wrapper_pos):
            wrapper_pos = backtick_first
            wrapper_token = "```"

        if wrapper_pos is not None:
            start = wrapper_pos
            end = s.find(wrapper_token, start + len(wrapper_token))
            if end != -1:
                inner = s[start + len(wrapper_token) : end]
                remainder = s[end + len(wrapper_token) :].strip()

                # if inner begins with stray ' or stray message marker,
remove it
                inner = inner.lstrip()
                inner = re.sub(r"^'+", "", inner)  # remove extra leading
single quotes
```

```
                inner =
re.sub(r"^\s*(?:'message'\s*:\s*|\"message\"\s*:\s*|message\s*:)\s*", "",
inner, flags=re.IGNORECASE)

                # extract meta from remainder (after the closing wrapper)
                speaker, score, gender, gender_conf =
_extract_meta(remainder)

                # if not found in remainder, maybe metadata sits after
additional separators like " : 'username':Name"
                if not speaker:
                    # search whole original string after wrapper for
'username' anywhere
                    m_all =
re.search(r"'username'\s*:\s*['\"]?(?P<u>[^'\"\n:]+)['\"]?",
s[end+len(wrapper_token):])
                    if m_all:
                        speaker = m_all.group("u").strip()

                # final message is inner (trim)
                message = inner.strip()

                return message, (speaker if speaker else None), (score if
score else None), (gender if gender else None), (gender_conf if
gender_conf else None)
            # if no closing wrapper found, fall through to normal parsing

        # 3) Try structured-string parsing with quoted keys (safer)
        try:
            pattern =
r"'message'\s*:\s*'(?P<message>.*?)'\s*:\s*'username'\s*:\s*'(?P<username
>[^']*)'(?:\s*:\s*'score'\s*:\s*'(?P<score>[^']*)')?(?:\s*:\s*'gender'\s*
:\s*'(?P<gender>[^']*)')?(?:\s*:\s*'gender_conf'\s*:\s*'(?P<gender_conf>[
^']*)')?"
            m = re.search(pattern, s, flags=re.DOTALL)
            if m:
                message_ = m.group('message').strip()
                speaker_ = m.group('username').strip() or None
                score_ = m.group('score') or None
                gender_ = m.group('gender') or None
                gender_conf_ = m.group('gender_conf') or None
                return message_, speaker_, score_, gender_, gender_conf_
        except Exception:
            pass

        # 4) Try to extract "'username':Name" anywhere (and remove it
from message candidate)
        m_user_any =
re.search(r"'username'\s*:\s*['\"]?(?P<u>[^'\"\n:]+)['\"]?", s)
        speaker = None
        score = None
        gender = None
        gender_conf = None
        if m_user_any:
```

```
            speaker = m_user_any.group("u").strip()
            # remove the matched token from s
            s = (s[: m_user_any.start()] + s[m_user_any.end():]).strip("
:\n\t ")

            # also attempt score/gender after removing username token
            ms =
re.search(r"'score'\s*:\s*['\"]?(?P<s>[^'\"\s:]+)['\"]?", s,
flags=re.IGNORECASE)
            if ms:
                score = ms.group("s").strip()
            mg =
re.search(r"'gender'\s*:\s*['\"]?(?P<g>[^'\"\s:]+)['\"]?", s,
flags=re.IGNORECASE)
            if mg:
                gender = mg.group("g").strip()
            mgc =
re.search(r"'gender_conf'\s*:\s*['\"]?(?P<gc>[^'\"\s:]+)['\"]?", s,
flags=re.IGNORECASE)
            if mgc:
                gender_conf = mgc.group("gc").strip()

            # remaining s is candidate message
            candidate = s.strip()
            return candidate, (speaker if speaker else None), (score if
score else None), (gender if gender else None), (gender_conf if
gender_conf else None)

        # 5) Try "username: Name" or "user: Name" at end
        m_user2 =
re.search(r"(?:\busername\b|\buser\b)\s*[:=]\s*['\"]?(?P<u>[^'\"\n]+)['\"
]?\s*$", s, flags=re.IGNORECASE)
        if m_user2:
            speaker = m_user2.group("u").strip()
            message_candidate =
re.sub(r"(?:[:\s]*\busername\b\s*[:=]\s*['\"]?.+?['\"]?\s*$)|(?:[:\s]*\bu
ser\b\s*[:=]\s*['\"]?.+?['\"]?\s*$)", "", s, flags=re.IGNORECASE).strip("
:")
            return message_candidate, (speaker if speaker else None),
None, None, None

        # 6) Last-token username heuristic "body : Name"
        m_user3 =
re.match(r"^(?P<body>.*\S)\s*:\s*(?P<u>[^\n:]{1,48})\s*$", s)
        if m_user3:
            maybe_u = m_user3.group("u").strip()
            maybe_body = m_user3.group("body").strip()
            if " " not in maybe_u or len(maybe_u) <= 24:
                return maybe_body, maybe_u, None, None, None

        # 7) fallback: return raw string (no username)
        return s, None, None, None, None

    # --- fallback for other types ---
```

```
    return str(query).strip(), None, None, None, None

# ----------------- Reminder MESSAGE extraction -------------------------
-------

from typing import Optional, Tuple

import ast
from typing import Optional, Tuple

def extract_text_from_reminder(query) -> Tuple[str, Optional[str],
Optional[float], Optional[str], Optional[float]]:
    """
    Returns: (message, speaker, score, gender, gender_conf)

    Accepts:
      - dict with keys like 'text'/'message'/'query' and
'speaker'/'username'/'user'
      - string containing a python-dict-like representation (will try
ast.literal_eval)
      - plain string (falls back to heuristics)
    """
    if query is None:
        return "", None, None, None, None

    # If it's already a dict-like object -> normalize and return
    if isinstance(query, dict):
        text_ = query.get("text") or query.get("query") or
query.get("message") or query.get("q") or ""
        speaker_ = query.get("username") or query.get("speaker") or
query.get("user")
        score_ = query.get("score")
        gender_ = query.get("gender")
        gender_conf_ = query.get("gender_conf")

        # normalize speaker (handle "None Home" / "None")
        if isinstance(speaker_, str):
            s = speaker_.strip()
            if s.lower().startswith("none "):
                s = s[5:].strip()
            if s.lower() == "none":
                speaker_ = None
            else:
                speaker_ = s

        # convert numeric-ish fields to floats where possible
        try:
            score_f = float(score_) if score_ is not None else None
        except Exception:
            score_f = None
        try:
            gconf_f = float(gender_conf_) if gender_conf_ is not None
else None
        except Exception:
```

```
            gconf_f = None

        return str(text_).strip(), (str(speaker_).strip() if speaker_ is
not None else None), score_f, (str(gender_) if gender_ is not None else
None), gconf_f

    # If it's a string, attempt to parse dict-like content first,
otherwise run heuristics
    if isinstance(query, str):
        s = query.strip()

        # Try base64 decode if looks like base64 (keeps your previous
behavior)
        try:
            if len(s) > 50 and re.fullmatch(r'[A-Za-z0-9+/=\s]+', s) and
'\n' not in s:
                try:
                    decoded = base64.b64decode(s).decode('utf-8')
                    if decoded:
                        s = decoded.strip()
                except Exception:
                    pass
        except Exception:
            pass

        # If it looks like a python-dict string, try literal_eval ->
recurse
        looks_like_dict = s.startswith("{") and ("'text'" in s or
'"text"' in s or "'speaker'" in s or '"speaker"' in s)
        if looks_like_dict:
            try:
                parsed = ast.literal_eval(s)
                if isinstance(parsed, dict):
                    return extract_text_from_query(parsed)
            except Exception:
                # try JSON parse as fallback
                try:
                    import json
                    parsed = json.loads(s)
                    if isinstance(parsed, dict):
                        return extract_text_from_query(parsed)
                except Exception:
                    pass

        # Heuristics fallback: attempt to extract common fields from a
string
        # extract text/message if present
        m_text =
re.search(r"(?:'text'|\"text\"|text)\s*[:=]\s*['\"](?P<t>.*?)['\"]", s)
        text_val = m_text.group("t").strip() if m_text else s

        # speaker / username / user
```

```
        m_speaker =
re.search(r"(?:'speaker'|\"speaker\"|username|user)\s*[:=]\s*['\"]?(?P<s>
[^'\"\n,}]+)['\"]?", s)
        speaker_val = m_speaker.group("s").strip() if m_speaker else None
        if speaker_val:
            if speaker_val.lower().startswith("none "):
                speaker_val = speaker_val[5:].strip()
            if speaker_val.lower() == "none":
                speaker_val = None

        # numeric fields
        m_score = re.search(r"'score'\s*[:=]\s*([0-9]*\.?[0-9]+)", s) or
re.search(r'"score"\s*[:=]\s*([0-9]*\.?[0-9]+)', s)
        score_val = float(m_score.group(1)) if m_score else None

        m_gender =
re.search(r"'gender'\s*[:=]\s*['\"]?(?P<g>[^'\"\n,}]+)['\"]?", s) or
re.search(r'"gender"\s*[:=]\s*["]?(?P<g>[^"]+)["]?', s)
        gender_val = m_gender.group("g").strip() if m_gender else None

        m_gconf = re.search(r"'gender_conf'\s*[:=]\s*([0-9]*\.?[0-9]+)",
s) or re.search(r'"gender_conf"\s*[:=]\s*([0-9]*\.?[0-9]+)', s)
        gender_conf_val = float(m_gconf.group(1)) if m_gconf else None

        return text_val, speaker_val, score_val, gender_val,
gender_conf_val

    # final fallback for other types
    return str(query).strip(), None, None, None, None

# ----------------- Reminder handlers  -----------------
import os
import time
##from datetime import datetime, timedelta
from datetime import timedelta
# helper stop keywords
_STOP_KEYWORDS = ("finished", "completed", "i am done", "thank you",
"stop", "complete", "that's all", "done")

# where we store simple reminders (plain text fallback)
_REMINDERS_PATH = os.path.join(os.getcwd(), "data.txt")

NUM_WORDS = {

"zero":0,"one":1,"two":2,"three":3,"four":4,"five":5,"six":6,"seven":7,"e
ight":8,"nine":9,

"ten":10,"eleven":11,"twelve":12,"thirteen":13,"fourteen":14,"fifteen":15
,"sixteen":16,

"seventeen":17,"eighteen":18,"nineteen":19,"twenty":20,"thirty":30,"forty
":40,"fifty":50,
    "sixty":60,"seventy":70,"eighty":80,"ninety":90
}
```

```
SCALES = {"hundred":100,"thousand":1000,"million":1_000_000}

def words_to_number(words):
    total = 0
    current = 0
    for w in words.split():
        if w in NUM_WORDS:
            current += NUM_WORDS[w]
        elif w in SCALES:
            current *= SCALES[w]
            if SCALES[w] > 100:
                total += current
                current = 0
        else:
            return None
    return total + current

import re
from word2number import w2n

def words_to_numbers_in_text(text: str) -> str:
    """
    Converts spoken numbers in a sentence to digits.
    Example:
      "remind me in three days at half past six"
      -> "remind me in 3 days at 6:30"
    """
    text = text.lower()

    # common spoken time fixes
    text = text.replace("half past", "30 ")
    text = text.replace("quarter past", "15 ")
    text = text.replace("quarter to", "45 ")

    tokens = text.split()
    out = []
    buffer = []

    def flush():
        nonlocal buffer
        if buffer:
            try:
                num = w2n.word_to_num(" ".join(buffer))
                out.append(str(num))
            except Exception:
                out.extend(buffer)
            buffer = []

    for t in tokens:
        if re.fullmatch(r"[a-z\-]+", t):
            buffer.append(t)
        else:
            flush()
            out.append(t)
```

```
    flush()
    return " ".join(out)

def _safe_setup_vosk_and_tts():
    vosk_model = None
    tts = None
    try:
        vosk_model = setup_vosk_model()
    except Exception:
        pass
    try:
        tts = setup_text_to_speech()
    except Exception:
        pass
    return vosk_model, tts

def _recognize_once(vosk_model=None, max_wait=15):
    """
    Try VOSK (if available) via recognize_speech +
listen_to_audio_Reminder().
    Fallback to listen.listen() if anything fails.
    Returns a str (may be empty).
    """
    try:
        if vosk_model is not None and 'listen_to_audio_Reminder' in
globals() and 'recognize_speech' in globals():
            audio = listen_to_audio_Reminder()
            message = recognize_speech(vosk_model, audio) or ""
            return str(message).strip()
    except Exception:
        pass

    # fallback: use your generic listen.listen()
    try:
        txt = listen.listen()
        return str(txt).strip() if txt else ""
    except Exception:
        # final fallback: no input available
        return ""

def _collect_lines_interactive(prompt: str, max_seconds: int = 90):
    """
    Speak the prompt then collect lines until user says a stop keyword or
max_seconds passes.
    Returns list[str] of collected lines (no stop keyword).
    """
    try:
        speech.AlfredSpeak(prompt)
    except Exception:
        print("ALFRED (speak):", prompt)
    try:
        listen.send_bluetooth(prompt)
    except Exception:
```

```
        pass

    vosk_model, _tts = _safe_setup_vosk_and_tts()

    lines = []
    start = time.time()
    silence_count = 0

    while True:
        # enforce overall timeout
        if time.time() - start > max_seconds:
            break

        recognized = _recognize_once(vosk_model)
        print(f"recognized     : {recognized}")

        # Example usage
        message, speaker, score, gender, gender_conf =
extract_text_from_reminder(recognized)

        New_Message = message
        New_speaker = speaker
        New_score = score
        New_gender = gender
        New_gender_conf = gender_conf

        print(f"New_Message Reminder     : {New_Message}")
        print(f"New_speaker Reminder     : {New_speaker}")
        print(f"New_score Reminder       : {New_score}")
        print(f"New_gender Reminder      : {New_gender}")
        print(f"New_gender_conf Reminder : {New_gender_conf}")

        if not recognized:
            # small idle wait to avoid busy loop
            time.sleep(0.6)
            silence_count += 1
            # if silence for some iterations, break to avoid hang
            if silence_count > 12:
                break
            continue

        silence_count = 0
        print("Collected voice:", New_Message)

        word2number_message = words_to_numbers_in_text(New_Message)
        print("Collected voice word2number_message:",
word2number_message)

        try:
            speech.AlfredSpeak(word2number_message)
        except Exception:
            pass

        normalized = word2number_message.strip()
```

```
        # if user said a stop phrase, stop collecting (do NOT include the
stop phrase)
        if any(k in normalized.lower() for k in _STOP_KEYWORDS):
            break

        lines.append(normalized)

    return lines

def _append_lines_to_file(lines, path=_REMINDERS_PATH):
    if not lines:
        return
    try:
        with open(path, "a", encoding="utf-8") as f:
            for ln in lines:
                f.write(ln.rstrip() + "\n")
        print(f"Saved {len(lines)} reminder lines to {path}")
    except Exception as e:
        print("Failed to save reminders:", e)

def _read_all_reminders(path=_REMINDERS_PATH):
    try:
        if not os.path.exists(path):
            return []
        with open(path, "r", encoding="utf-8") as f:
            content = f.read().strip().splitlines()
        return [c for c in content if c.strip()]
    except Exception as e:
        print("Failed to read reminders:", e)
        return []

def _simple_write_ics(title: str, dtstart: datetime, duration_minutes:
int = 60) -> str:
    """
    Creates a very simple .ics file next to data.txt and returns the
path.
    If you added the organizer helper with add_organizer_event, prefer
that instead.
    """
    try:
        from uuid import uuid4
        uid = uuid4().hex
        dtend = dtstart + timedelta(minutes=int(duration_minutes))
        ics_lines = [
            "BEGIN:VCALENDAR",
            "PRODID:-//Alfred//EN",
            "VERSION:2.0",
            "CALSCALE:GREGORIAN",
            "BEGIN:VEVENT",
            f"UID:{uid}",
            f"DTSTAMP:{datetime.utcnow().strftime('%Y%m%dT%H%M%SZ')}",
            f"DTSTART:{dtstart.strftime('%Y%m%dT%H%M%S')}",
            f"DTEND:{dtend.strftime('%Y%m%dT%H%M%S')}",
            f"SUMMARY:{title}",
```

```
            "END:VEVENT",
            "END:VCALENDAR",
            ""
        ]
        ics_path = os.path.join(os.getcwd(), f"reminder_{uid}.ics")
        with open(ics_path, "w", encoding="utf-8", newline="\r\n") as f:
            f.write("\r\n".join(ics_lines))
        return ics_path
    except Exception as e:
        print("Failed to write ics:", e)
        return ""



def main(assistant, gui):

    while True:

        # inside while True:
        global query_queue, query_queue_then, task_index,
processing_task, AlfredQueryOffline, and_then_speaker
        global _last_activity_time, _standby_triggered, _standby_running

        # Respect app shutdown signal
        if app_shutdown_event.is_set():
            print("main(): shutdown event received, exiting loop.")
            break

        try:
            queued = query_queue.get_nowait()  # non-blocking
            query = queued
            print("[query] GOT from query_queue :", query)

        except queue.Empty:
            # No queued command — decide whether to call the mic
listener.
            # Avoid calling listen.listen() while TTS is active, paused,
or when
            # some other part of the app set _suppress_auto_listen.
            try:
                # --- 1) Is speech currently speaking? ---
                speaking = False
                try:
                    # prefer public method
                    is_speaking_fn = getattr(speech, "is_speaking", None)
                    if callable(is_speaking_fn):
                        speaking = bool(is_speaking_fn())
                    else:
                        # fallback to internal flag
                        speaking = bool(getattr(speech,
"_currently_speaking", False))

                    # if not yet detected, check any player object safely
```

```
                    if not speaking:
                        player = getattr(speech, "_player", None)
                        if player is not None:
                            try:
                                # prefer is_playing() if available
                                is_playing = getattr(player,
"is_playing", None)
                                if callable(is_playing):
                                    speaking = bool(is_playing())
                                else:
                                    # try get_state() if available and
compare to vlc.State when possible
                                    get_state = getattr(player,
"get_state", None)
                                    if callable(get_state):
                                        st = get_state()
                                        try:
                                            import vlc as _vlc  # only
try if available on system
                                            speaking = (st ==
_vlc.State.Playing)
                                        except Exception:
                                            # best-effort: treat certain
named states as not playing
                                            try:
                                                sname = str(st).lower()
                                                speaking = not (sname in
("stopped", "ended", "error", "none"))
                                            except Exception:
                                                # if we cannot determine,
leave speaking as False
                                                pass
                            except Exception:
                                # ignore player introspection errors
                                pass
                except Exception:
                    speaking = False

                # --- 2) Is speech paused? ---
                paused = False
                try:
                    is_paused_fn = getattr(speech, "is_paused", None)
                    if callable(is_paused_fn):
                        paused = bool(is_paused_fn())
                    else:
                        paused = bool(getattr(speech, "_pause_requested",
False) or (getattr(speech, "_paused_file", None) is not None))
                except Exception:
                    paused = False

                # --- 3) Is auto-listen suppressed explicitly? ---
                suppress_listen = False
                try:
```

```
                    suppress_listen = bool(getattr(speech,
"_suppress_auto_listen", False))
                except Exception:
                    suppress_listen = False

                # If any reason to avoid starting the mic, skip calling
listen.listen()

##                if speaking or paused or suppress_listen:
                if speaking or paused or suppress_listen or
get_reminder_speaking_flag():
                    try:
##                        log_queue.put(f"skip listen():
speaking={speaking} paused={paused} suppressed={suppress_listen}")
                        log_queue.put(f"skip listen():
speaking={speaking} paused={paused} suppressed={suppress_listen}
reminder_speaking={get_reminder_speaking_flag()}")
                    except Exception:
                        pass
                    # small sleep to avoid busy-spin while remaining
responsive
                    time.sleep(0.12)
                    continue

                # safe to call the microphone listener
                try:
                    query = listen.listen()
                except Exception as e:
                    print("□ listen.listen() error:", e)
                    query = None

                print("[query] in main (mic) :", query)

            except Exception as e_outer:
                # Catch-all to avoid the main loop dying due to
unexpected errors here
                print("Unexpected error while deciding to listen:",
e_outer)
                time.sleep(0.12)
                continue

        import re
        import base64

        # Example usage
        message, speaker, score, gender, gender_conf =
extract_text_from_query(query)

        New_Message = message
        New_speaker = speaker
        New_score = score
        New_gender = gender
        New_gender_conf = gender_conf
```

```
print(f"New_Message     : {New_Message}")
print(f"New_speaker     : {New_speaker}")
print(f"New_score       : {New_score}")
print(f"New_gender      : {New_gender}")
print(f"New_gender_conf : {New_gender_conf}")

if New_speaker is None:
    New_speaker = ""

if speaker is None:
    speaker = ""

try:
    if ("Celinda" or "Selena" or "Dalinya") in New_speaker:
        New_gender_call = "Madam"
    else:

        if "Tjaart" or "Sebastiaan" in New_speaker:
            New_gender_call = "Master"

except:
    if "Male" in  New_gender:
        New_gender_call = "Sir"
    elif "Female" in  New_gender:
        New_gender_call = "Missus"
    continue

username = speaker

# default speaker if missing (you used this later)
if New_speaker is None:
    New_speaker = "ITF"

if speaker is not None:
    speech_sure_name = speaker
    speech_sure_name = speech_sure_name.replace(" Home","")
    speech_sure_name = speech_sure_name.replace(" (Male)","")
    speech_sure_name = speech_sure_name.replace(" (Female)","")
    print(f"speech_sure_name   : {speech_sure_name}")

if speech_sure_name is not None:
    set_current_speaker(speech_sure_name)

if speech_sure_name is None:
    speech_sure_name == ""

speaker_speak = New_speaker


#----  Standby Timer  ----

now = time.monotonic()
```

```
        # New_Message should be a string when there is user input, or
falsy (None/''/False) when no new input.
        if New_Message:
            # any activity resets the timer
            _last_activity_time = now

            # if standby is running, stop it (user became active again)
            if _standby_running:
                try:
                    alfred.stop_service()
                except Exception as e:
                    print("[standby] stop_service error:", e)
                _standby_running = False

            # if the incoming message contains a wake word while standby
was not running,
            # you can handle it normally here (example shown)
            # if you specifically want to cancel standby on certain
words, also check here:
            lower_msg = (New_Message or "").lower()
            if any(w in lower_msg for w in WAKE_WORDS_STANDBY):
                # do any immediate reaction if needed
                # already stopped above, because there was activity
                pass

        else:
            # no new message -> check idle time
            idle_time = now - _last_activity_time
            # debug print
            print(f"[DEBUG idle_time] idle_time = {idle_time}")

            # if we've been idle long enough and standby isn't already
running -> start it once
            if idle_time >= STANDBY_TIMEOUT_SECONDS and not
_standby_running:
                try:
                    # start the standby service once
                    alfred.start_service(background=True,
use_camera_input=Alfred_config.CHEST_CAMERA_INPUT,
enable_breathing=False, enable_jitter=False)
                    _standby_running = True
                    print("[standby] started (idle timeout reached)")
                except Exception as e:
                    print("[standby] start_service() failed:", e)

        # sleep or continue loop as your main program requires
        time.sleep(0.1)

        # --- DEBUG current queue state ---
        print(f"[DEBUG] queue_len={len(query_queue_then)}
task_index={task_index} processing_task={processing_task}")

        # If the incoming message contains "and then" and we're not
already stepping a queue,
```

```
        # build the structured queue exactly as requested
        if message and ("and then" in message or "after that" in message)
and not processing_task:

            and_then_speaker = speaker_speak

            MyToDoListEdited = [t.strip() for t in message.split(" and
then " or " after that ") if t.strip()]
            print("\nMy things ask to do list :", MyToDoListEdited, "\n")

            ToDoList.insert(1, message)
            ToDoListEdited = " ".join(dict.fromkeys(message.split()))
            print("\nThing ask to do list :", ToDoList, "\n")
            print("Thing do list edited:", ToDoListEdited, "\n")

            ThingsToDo = "\n".join(ToDoList)
            AlfredQueryOfflineToDoList = "\n" + ThingsToDo
            print("\nThings going to do :", AlfredQueryOfflineToDoList,
"\n")
            print("My things ask to do list Next 2 :", MyToDoListEdited,
"\n")

            # --- Build structured task dicts (ALWAYS this structure) ---
            query_queue_then = [
                {"message": task, "username": and_then_speaker, "score":
None}
                for task in MyToDoListEdited
            ]

            task_index = 0
            processing_task = True
            print("Query queue structured :", query_queue_then, "\n")

        # Decide what AlfredQueryOffline should be for THIS iteration.
        # Priority:
        # 1) If we have a fresh spoken/GUI message (non-empty), use it
as-is.
        # 2) Else, if we are processing a queued set, pull the next task.
        # 3) Else, nothing to do this loop.

        picked_from_queue = False
        if message:
            AlfredQueryOffline = message
        else:
            # No fresh speech — try to advance the queue
            if processing_task and query_queue_then and task_index <
len(query_queue_then):
                current_task = query_queue_then[task_index]
                AlfredQueryOffline = current_task["message"]  # ALWAYS a
string
                picked_from_queue = True
```

```
##                AlfredQueryOffline_Text =
f"'message':{AlfredQueryOffline} : 'username':{and_then_speaker} :
'score':None"
                AlfredQueryOffline_Text = f"{AlfredQueryOffline}"
                AlfredQueryOffline = AlfredQueryOffline_Text

                print(f"[Task #{task_index+1}/{len(query_queue_then)}]
:", AlfredQueryOffline)
                print("[My_new_AND_Prompt_and_then] :",
AlfredQueryOffline)
                task_index += 1
            else:
                # If the queue is finished, reset state
                if processing_task and (not query_queue_then or
task_index >= len(query_queue_then)):
                    print("[All tasks completed]")
                    query_queue_then = []
                    task_index = 0
                    processing_task = False

                    speaker_speak = and_then_speaker
                AlfredQueryOffline = ""  # nothing to process this loop

        # If still empty, skip doing work this iteration
        if not AlfredQueryOffline:
            print(f" THERE IS NO AlfredQueryOffline:
{AlfredQueryOffline}")
            continue

        # If this was a fresh message that did NOT contain "and then" but
we still have an
        # active queue from a previous turn, you can choose to continue
the queue *after*
        # responding to this fresh message, or cancel it. Current logic
keeps the queue
        # and will continue auto-advancing on subsequent empty
iterations.

        print(f" Text: {AlfredQueryOffline}")
        print(f" Speaker: {speaker_speak}")
        print(f" Score: {score}")
        memory.add_to_memory({"timestamp": time.time(), "text":
AlfredQueryOffline, "speaker": speaker_speak})

        print(f" You said: {query}")
        print(f"You said AlfredQueryOffline : {AlfredQueryOffline}")

        # Continue with your downstream processing
        AlfredQueryOffline_Doing = AlfredQueryOffline
        print(f"You said AlfredQueryOffline_Doing :
{AlfredQueryOffline_Doing}")

        # Debug print for visibility each loop
```

```
        print(f"[DEBUG] queue_len={len(query_queue_then)}
task_index={task_index} processing_task={processing_task}")

       # Now carry on with the rest of the processing that uses
AlfredQueryOffline_Doing etc.
        AlfredQueryOffline_Doing = AlfredQueryOffline
        print(f"You said AlfredQueryOffline_Doing :
{AlfredQueryOffline_Doing}")

        AlfredQueryOffline_Doing = AlfredQueryOffline

        print(f"You said AlfredQueryOffline_Doing :
{AlfredQueryOffline_Doing}")

        Alfred_No_Ollama = 1

        AlfredQueryOffline = AlfredQueryOffline.replace("i'm going to go
ahead and get a little bit of a little bit of a little bit","")
        AlfredQueryOffline = AlfredQueryOffline.replace("i'm going to go
ahead and get a little bit of a", "")
        # Step 1: normalize spaces
        cleaned = re.sub(r'\s+', ' ', AlfredQueryOffline).strip()

        words = cleaned.split()
        unique_words = []
        for w in words:
            if not unique_words or w != unique_words[-1]:  # skip direct
repeats
                unique_words.append(w)

        result = " ".join(unique_words)
        print(result)

        AlfredQueryOffline = result
        print(f"Cleaned without double sentances : {AlfredQueryOffline}")

        if ('talking to you' in AlfredQueryOffline or 'who is speaking'
in AlfredQueryOffline or 'who is talking' in AlfredQueryOffline
               or 'who am i' in AlfredQueryOffline):

            My_Message_Doing = AlfredQueryOffline

            My_Message_Did = AlfredQueryOffline

            NewPromptEdited_Did = My_Message_Did

            NewPromptEdited_Did = NewPromptEdited_Did.replace("who is",
"who was")
            NewPromptEdited_Did = NewPromptEdited_Did.replace("can you",
"")
            NewPromptEdited_Did = NewPromptEdited_Did.replace("who am i",
"you are")
            NewPromptEdited_Did = NewPromptEdited_Did.replace("what is my
name", "your name is")
```

```
            NewPromptEdited_Did = NewPromptEdited_Did.replace("talking to
you", "talking to me")
            NewPromptEdited_Did = NewPromptEdited_Did.replace("speaking
to you", "speaking to me")
            NewPromptEdited_Did = NewPromptEdited_Did.replace("please",
"")
            NewPromptEdited_Did = NewPromptEdited_Did.replace("tell me ",
"I told you ")

            New_Prompt_Speaking = NewPromptEdited_Did

            print(f"NewPromptEdited_Did: {NewPromptEdited_Did}
{speaker_speak}")

            Did_History.insert(0, f"{NewPromptEdited_Did}
{speaker_speak}")
            Did_History.insert(1, f". and . ")

            print('sir: ' + AlfredQueryOffline)
            Alfred_No_Ollama = 0
            print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)

            speech.AlfredSpeak(f"{New_Prompt_Speaking} {speaker_speak}")
            AlfredQueryOffline = ""

        # Existing processing logic
        try:
            if AlfredQueryOffline and len(AlfredQueryOffline) >= 15:
                Alfred_No_Ollama = 1
                print(f"Sure : {New_gender_call} {speech_sure_name}")
                speech.AlfredSpeak(f"Sure : {New_gender_call}
{speech_sure_name}")
                current_date = datetime.datetime.now().strftime('%Y-%m-
%d')
                current_time =
datetime.datetime.now().strftime('%H:%M:%S')


##                query_msg = f"{current_date} : {current_time} :
{AlfredQueryOffline} : 'username':{speaker_speak}"
                query_msg = f"{current_date} : {current_time} :
{AlfredQueryOffline} : {speaker_speak}"

                # Use the gui instance passed into main() to log; handle
None or exceptions gracefully
                if gui is not None:
                    try:
                        gui.log_query(query_msg)
                    except Exception as e:
                        print("Error logging to GUI instance:", e)
                else:
                    print("GUI instance not available for logging message
(gui is None).")
```

```
                AlfredQueryOffline = query_msg

                print(f"\n FINAL AlfredQueryOffline for ASSISTANT QUERY
GOING FOR PROCESSING IS : {AlfredQueryOffline} \n")

                print(f"\n \n
########################################################################
\n \n") #``` This code will output: ``` james is bad ```

                # --- Extract text inside triple single quotes ---
                pattern = r"'''(.*?)'''"
                match = re.search(pattern, AlfredQueryOffline, re.DOTALL)
                if match:
                    extracted_text = match.group(1).strip()
                else:
                    extracted_text = AlfredQueryOffline.strip()  #
fallback to full text if no triple quotes found

                print(f"\n\n extracted_text : {extracted_text} \n\n")

                # --- Split on the first question mark ---
                parts = re.split(r'\?', extracted_text, maxsplit=1)

                if len(parts) == 2:
                    question_asked = parts[0].strip() + '?'
                    context_added = parts[1].strip()
                else:
                    question_asked = extracted_text
                    context_added = ""

                print(f"\nquestion_asked : {question_asked}\n")
                print(f"context_added  : {context_added}\n")

                match = re.match(r"^\d{4}-\d{2}-
\d{2}\s*:\s*\d{2}:\d{2}:\d{2}\s*:\s*(.*)", question_asked)
                updated_query = match.group(1).strip() if match else
question_asked.strip()

                # Remove the USERNAME part from the query
                final_updated_query = updated_query.split(":",
1)[0].strip()


                print(f"\n \n ###---------------
############################----------########## \n \n") #``` This
code will output: ``` james is bad ```
                print(f"\n final_updated_query :
{final_updated_query}\n")
                print(f"\n \n ###---------------
############################----------########## \n \n") #``` This
code will output: ``` james is bad ```
```

```
                print(f"\n \n
##########################################################################
\n \n") #``` This code will output: ``` james is bad ```

                AlfredQueryOffline = question_asked
                print(f"\n AlfredQueryOffline : {AlfredQueryOffline}\n")

                AlfredQueryOffline_assistant = final_updated_query
                print(f"\n AlfredQueryOffline_assistant :
{AlfredQueryOffline_assistant}\n")

##                AlfredQueryOffline_assistant = extracted_text
##                print(f"\n AlfredQueryOffline_assistant :
{AlfredQueryOffline_assistant}\n")

                print(f"\n \n
##########################################################################
\n \n") #``` This code will output: ``` james is bad ```

#--------------------------------------------------------------------------
-------------
#-------------------------  REMINDERS & TIMED COMMANDS  -------------------
-------------------------

                import traceback
                import re

                # --- your unchanged trigger lists, kept as-is but fix
the tuple typo for exclusions ----
                reminder_triggers = (
                    "remind me", "create a reminder", "create me a
reminder",
                    "set a reminder", "set reminder", "i want to
remember",
                    "can you remember", "what can you remember", "what do
i have",
                    "what are my reminders", "what is my schedule", "what
do i have scheduled",
                    "what did i tell you", "what are my meetings",
"schedule me a meeting with",
                    "create me a meeting", "setup a meeting"
                )

                command_time_pattern = (
                    "clock", "oclock", "half past", "quarter past",
"quarter to",
                    "minutes", "minute", "hours", "hour", "seconds",
"second", "tomorrow at", "today at", "at noon", "at midnight",
                    "o'clock", "o clock", "oclock", "o'clock", "o clock",
"oclock"
                )

                timed_command_number = (
                    # words
```

```
                    "one", "two", "three", "four", "five", "six",
"seven", "eight", "nine", "ten",
                    "eleven", "twelve", "thirteen", "fourteen",
"fifteen", "sixteen", "seventeen",
                    "eighteen", "nineteen",
                    "twenty", "twenty one", "twenty two", "twenty three",
"twenty four",
                    "twenty five", "twenty six", "twenty seven", "twenty
eight", "twenty nine",
                    "thirty", "thirty one", "thirty two", "thirty three",
"thirty four",
                    "thirty five", "thirty six", "thirty seven", "thirty
eight", "thirty nine",
                    "forty", "forty one", "forty two", "forty three",
"forty four",
                    "forty five", "forty six", "forty seven", "forty
eight", "forty nine",
                    "fifty", "fifty one", "fifty two", "fifty three",
"fifty four",
                    "fifty five", "fifty six", "fifty seven", "fifty
eight", "fifty nine",

                    # digits
                    "1", "2", "3", "4", "5", "6", "7", "8", "9", "10",
                    "11", "12", "13", "14", "15", "16", "17", "18", "19",
                    "20", "21", "22", "23", "24", "25", "26", "27", "28",
"29",
                    "30", "31", "32", "33", "34", "35", "36", "37", "38",
"39",
                    "40", "41", "42", "43", "44", "45", "46", "47", "48",
"49",
                    "50", "51", "52", "53", "54", "55", "56", "57", "58",
"59"
                )

                # Make this a single-element tuple (was a string before:
("until tomorrow") -> "u...")
                reminder_triggers_exclusions = (
                    "until tomorrow",
                )

                # ---------- Main check (minimal edits) ----------
                # Normalise query text and pick the correct source
variable:
                # prefer AlfredQueryOffline_assistant if present, then
AlfredQueryOffline, else empty string.
                query_text_raw = AlfredQueryOffline_assistant if
'AlfredQueryOffline_assistant' in globals() and
AlfredQueryOffline_assistant else (AlfredQueryOffline if
'AlfredQueryOffline' in globals() and AlfredQueryOffline else "")
                # lowercased form for safe substring checks
                query_text = query_text_raw.lower()
```

```
                # (Optional) you can strip punctuation if you experience
false positives inside words:
                # query_text = re.sub(r"[^\w\s']", " ", query_text)  #
keep apostrophes if needed

                if ((any(word in query_text for word in
reminder_triggers)
                     or any(word in query_text for word in
command_time_pattern))
                    and any(word in query_text for word in
timed_command_number)
                    and not any(exc in query_text for exc in
reminder_triggers_exclusions)):
                    try:
                        # Only check delegation if there is text (guards
busy-loop)
                        if query_text_raw:
                            # pass the original raw text to delegation
functions (in case they rely on original formatting)
                            if handle_reminder_delegation(query_text_raw,
gui):
                                # reminder consumed it — go to next loop
iteration
                                continue
                            if
handle_timed_command_delegation(query_text_raw, gui):
                                # timed command consumed it (scheduled or
rejected) — next loop
                                continue
                        # else: no delegation matched -> fall through to
normal processing
                    except Exception as e:
                        print("Delegation call error:", e)
                        traceback.print_exc()
                        pass

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


                if "send" and "an email" in AlfredQueryOffline_assistant:

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)

                    speech.AlfredSpeak(f"Ok, let's set you up for a quick
email, sir")
                    time.sleep(1)
```

```
                    try :
                        if MyToDoListEdited != []:
                            MyToDoListEdited.pop(0)
                            AlfredQueryOfflineNew = MyToDoListEdited
                            print("")
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                        ##    print(AlfredQueryOfflineNew)
                            print('')
                    except:
                        pass

                    assistant.handle_send_email()

                # --- START OLLAMA ---
                elif ('start ollama server' in
AlfredQueryOffline_assistant or 'start ollama' in
AlfredQueryOffline_assistant
                      or 'launch ollama' in AlfredQueryOffline_assistant
or 'run ollama' in AlfredQueryOffline_assistant
                      or 'open ollama' in AlfredQueryOffline_assistant or
'bring up ollama' in AlfredQueryOffline_assistant
                      or 'start the ollama' in
AlfredQueryOffline_assistant):

                    My_Message_Doing = AlfredQueryOffline
                    My_Message_Did = AlfredQueryOffline
                    NewPromptEdited_Did = My_Message_Did

                    # same style of replacements as your other blocks
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("start", "starting")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)
                    try:
                        Did_History.insert(0, f"{NewPromptEdited_Did}")
```

```
                        Did_History.insert(1, f". and . ")
                    except Exception as e:
                        print("warning inserting Did_History:", e)

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ', Alfred_No_Ollama)

                    # announce to the user
                    try:
                        speech.AlfredSpeak("Starting Ollama server...")
                    except Exception:
                        pass

                    # working directory (fall back to cwd if 'dir' not
set)
                    work_dir = globals().get('dir', os.getcwd())

                    # Optional: path to a virtualenv activate.bat (set in
globals() if you need it)
                    venv_activate = globals().get('VENV_ACTIVATE', None)
# e.g. r"C:\path\to\venv\Scripts\activate.bat"

                    # Try to find the ollama executable path so the new
shell can run it even if PATH differs
                    ollama_path = shutil.which("ollama")
                    if ollama_path:
                        # quote it for safe use in batch
                        quoted_ollama = f'"{ollama_path}"'
                    else:
                        # fall back to plain "ollama" (batch will rely on
PATH or venv activation)
                        quoted_ollama = "ollama"

                    if os.name == 'nt':
                        # Windows: create a .bat that activates venv (if
provided) then runs `cmd /k "<ollama> serve"`
                        bat_path = os.path.join(work_dir,
"ollama_serve.bat")
                        try:
                            if not os.path.exists(bat_path):
                                with open(bat_path, "w", encoding="utf-
8") as f:
                                    f.write('@echo off\r\n')
                                    f.write(f'cd /d "{work_dir}"\r\n')
                                    if venv_activate:
                                        f.write(f'call
"{venv_activate}"\r\n')
                                    # run ollama serve — cmd /k keeps the
window open and shows output
                                    f.write(f'cmd /k {quoted_ollama}
serve\r\n')
```

```
                            # Launch the batch in a new console and keep
handle (so we can terminate it later)
                            try:
                                CREATE_NEW_CONSOLE = getattr(subprocess,
"CREATE_NEW_CONSOLE", 0x00000010)
                                proc = subprocess.Popen(["cmd", "/k",
bat_path],
                                                        cwd=work_dir,

creationflags=CREATE_NEW_CONSOLE)
                                globals()['OLLAMA_PROCESS'] = proc
                                print("Started ollama via batch (cmd /k),
pid=", getattr(proc, "pid", None))
                            except Exception as e:
                                print("Failed to launch cmd /k batch with
Popen:", e)
                                try:
                                    # fallback: open the batch by
os.startfile (no Popen handle)
                                    os.startfile(bat_path)
                                    print("Started ollama via
os.startfile (no handle).")
                                except Exception as e2:
                                    print("os.startfile failed:", e2)
                                    try:
                                        speech.AlfredSpeak("Could not
start Ollama using startfile.")
                                    except Exception:
                                        pass

                        except Exception as e:
                            print("Error preparing/starting ollama
batch:", e)
                            try:
                                speech.AlfredSpeak("Failed to start
Ollama server. Check logs.")
                            except Exception:
                                pass

                    else:
                        # POSIX: start the service directly using Popen
and put it in its own session
                        try:
                            # prefer absolute path found by shutil.which
if available
                            cmd = [ollama_path or "ollama", "serve"]
                            proc = subprocess.Popen(cmd, cwd=work_dir,
start_new_session=True,

stdout=subprocess.PIPE, stderr=subprocess.PIPE)
                            globals()['OLLAMA_PROCESS'] = proc
                            print("Started ollama (POSIX) with
subprocess.Popen, pid=", getattr(proc, "pid", None))
                        except Exception as e:
```

```
                            print("Failed to start ollama on POSIX:", e)
                            try:
                                speech.AlfredSpeak("Failed to start
Ollama server on this system.")
                            except Exception:
                                pass

                    # maintain the same MyToDoListEdited popping
behaviour you used elsewhere
                    try:
                        if MyToDoListEdited != []:
                            MyToDoListEdited.pop(0)
                            AlfredQueryOfflineNew = MyToDoListEdited
                            print("")
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                            print('')
                    except Exception:
                        pass


                # --- STOP OLLAMA ---
                elif ('stop ollama' in AlfredQueryOffline_assistant or
'shutdown ollama' in AlfredQueryOffline_assistant
                      or 'kill ollama' in AlfredQueryOffline_assistant or
'terminate ollama' in AlfredQueryOffline_assistant
                      or 'stop the ollama' in
AlfredQueryOffline_assistant or 'close ollama' in
AlfredQueryOffline_assistant or 'shut down ollama' in
AlfredQueryOffline_assistant):

                    My_Message_Doing = AlfredQueryOffline
                    My_Message_Did = AlfredQueryOffline
                    NewPromptEdited_Did = My_Message_Did

                    # same style replacements as your other blocks
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")
```

```
                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)
                    try:
                        Did_History.insert(0, f"{NewPromptEdited_Did}")
                        Did_History.insert(1, f". and . ")
                    except Exception as e:
                        print("warning inserting Did_History:", e)

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ', Alfred_No_Ollama)

                    try:
                        speech.AlfredSpeak("Stopping the Ollama
server...")
                    except Exception:
                        pass

                    stopped_any = False

                    # 1) If we have a process handle from this Python
session, try to terminate it cleanly
                    try:
                        proc = globals().get('OLLAMA_PROCESS', None)
                        if proc is not None:
                            try:
                                if proc.poll() is None:
                                    proc.terminate()
                                    try:
                                        proc.wait(timeout=5)
                                    except Exception:
                                        proc.kill()
                                        try:
                                            proc.wait(timeout=5)
                                        except Exception:
                                            pass
                                    stopped_any = True
                                    print("Stopped OLLAMA_PROCESS (via
Popen handle).")
                            except Exception as e:
                                print("Error terminating
OLLAMA_PROCESS:", e)
                            # clear global
                            try:
                                globals()['OLLAMA_PROCESS'] = None
                            except Exception:
                                pass
                    except Exception as e:
                        print("Error checking OLLAMA_PROCESS:", e)

                    # 2) OS-level attempt to kill any remaining 'ollama'
processes
                    try:
                        if os.name == 'nt':
```

```
                            try:
                                subprocess.run(["taskkill", "/IM",
"ollama.exe", "/F"], check=True,
                                               stdout=subprocess.PIPE,
stderr=subprocess.PIPE)
                                stopped_any = True
                                print("taskkill ollama.exe succeeded.")
                            except subprocess.CalledProcessError:
                                try:
                                    subprocess.run(["taskkill", "/IM",
"ollama", "/F"], check=True,

stdout=subprocess.PIPE, stderr=subprocess.PIPE)
                                    stopped_any = True
                                    print("taskkill ollama succeeded.")
                                except subprocess.CalledProcessError as
e:
                                    print("taskkill couldn't find or stop
ollama:", e)
                        else:
                            try:
                                subprocess.run(["pkill", "-f", "ollama
serve"], check=True,
                                               stdout=subprocess.PIPE,
stderr=subprocess.PIPE)
                                stopped_any = True
                                print("pkill -f 'ollama serve'
succeeded.")
                            except subprocess.CalledProcessError:
                                try:
                                    subprocess.run(["pkill", "ollama"],
check=True,

stdout=subprocess.PIPE, stderr=subprocess.PIPE)
                                    stopped_any = True
                                    print("pkill 'ollama' succeeded.")
                                except subprocess.CalledProcessError as
e:
                                    print("pkill couldn't find or stop
ollama:", e)
                    except Exception as e:
                        print("OS-level kill attempt failed:", e)

                    # final announce and housekeeping
                    if stopped_any:
                        print("Ollama server stopped.")
                        try:
                            speech.AlfredSpeak("Ollama server stopped.")
                        except Exception:
                            pass
                    else:
                        print("No Ollama process detected or stop
failed.")
                        try:
```

```
                            speech.AlfredSpeak("I could not find a
running Ollama server, or stopping failed. You may need to stop it
manually.")
                        except Exception:
                            pass

                    # keep your existing MyToDoListEdited pop behaviour
                    try:
                        if MyToDoListEdited != []:
                            MyToDoListEdited.pop(0)
                            AlfredQueryOfflineNew = MyToDoListEdited
                            print("")
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                            print('')
                    except Exception:
                        pass


                elif ("flash on" in AlfredQueryOffline_assistant):

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Doing = My_Message_Doing.replace("close",
"closing")

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("close", "closing")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
```

```
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)
                    assistant.set_flash_on()


                elif ("flash off" in AlfredQueryOffline_assistant):

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Doing = My_Message_Doing.replace("close",
"closing")

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("close", "closing")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)
                    assistant.set_flash_off()


                elif ("open door" in AlfredQueryOffline_assistant or
"open the door" in AlfredQueryOffline_assistant or 'open that door' in
AlfredQueryOffline_assistant
```

```
                    or 'open please' in AlfredQueryOffline_assistant):

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Doing = My_Message_Doing.replace("open",
"opening")

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("open", "opened")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)
                    SearchInput = AlfredQueryOffline

                    CheckLeftWord = 'search for'
                    CheckRightWord = 'please' or 'thank you' or 'thanks'
or 'ok'

                    index1 = SearchInput.find(CheckLeftWord)
                    index2 = SearchInput.find(CheckRightWord)

                    NewGoogleQuery = SearchInput[index1
+len(CheckLeftWord) + 1: index2]
                    print("NewGoogleQuery : " + NewGoogleQuery )
```

```
home_auto.Arduino_Home_Automation_Send_Door_Open(AlfredQueryOffline)

                elif ("close door" in AlfredQueryOffline_assistant or
"close the door" in AlfredQueryOffline_assistant or 'close that door' in
AlfredQueryOffline_assistant
                    or 'close please' in AlfredQueryOffline_assistant or
"shut the door" in AlfredQueryOffline_assistant or "shut lounge door" in
AlfredQueryOffline_assistant):

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Doing = My_Message_Doing.replace("close",
"closing")

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("close", "closing")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)

home_auto.Arduino_Home_Automation_Send_Door_Close(AlfredQueryOffline)


                elif ("switch off the lights" in
AlfredQueryOffline_assistant or 'switch the lights off' in
```

```
AlfredQueryOffline_assistant or 'off the lights' in
AlfredQueryOffline_assistant
                      or "lights off" in AlfredQueryOffline_assistant or
'off lights' in AlfredQueryOffline_assistant or 'sleep time' in
AlfredQueryOffline_assistant
                      or "turn off the lights" in
AlfredQueryOffline_assistant or "turn the lights off" in
AlfredQueryOffline_assistant or "turn off lights" in
AlfredQueryOffline_assistant   ):

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)
                    speech.AlfredSpeak("ok sir lets switch off the
lights")

home_auto.Arduino_Home_Automation_Lights_Off(AlfredQueryOffline)


                elif ("switch the lights on" in
AlfredQueryOffline_assistant or 'switch on the lights' in
AlfredQueryOffline_assistant or 'switch the lights on' in
AlfredQueryOffline_assistant
```

```
                    or "switch on lights" in
AlfredQueryOffline_assistant or 'lights on' in
AlfredQueryOffline_assistant or 'on lights' in
AlfredQueryOffline_assistant or 'switch on the light' in
AlfredQueryOffline_assistant
                    or "turn on the lights" in
AlfredQueryOffline_assistant or "turn the lights on" in
AlfredQueryOffline_assistant or "turn on lights" in
AlfredQueryOffline_assistant ):

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)
                    speech.AlfredSpeak("ok sir lets switch on the
lights")

home_auto.Arduino_Home_Automation_Lights_On(AlfredQueryOffline)


                elif ("check analog" in AlfredQueryOffline_assistant or
'check water' in AlfredQueryOffline_assistant
                    or 'check level' in AlfredQueryOffline_assistant or
'water level' in AlfredQueryOffline_assistant):
```

```
                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)

                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)


                    speech.AlfredSpeak("The water level control is not
yet enabled, Sir?")

                elif "am stress" in AlfredQueryOffline_assistant:

                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)


                    print('sir: ' + AlfredQueryOffline)
                    speech.AlfredSpeak(f'Here is a relaxing tune for you,
sir, hope you enjoy it')

os.startfile(Alfred_config.DRIVE_LETTER+'Python_Env//New_Virtual_Env//Pro
ject_Files//My_Mp3//chill songs.mp3')
                    wait()
                    speech.AlfredSpeak(f'What else would you like me to
do, sir?')
```

```
                    print('')
                    print('listening ...')
                    print('')

                    try :
                        if MyToDoListEdited != []:
                            MyToDoListEdited.pop(0)
                            AlfredQueryOfflineNew = MyToDoListEdited
                            print("")
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                            print('')
                    except:
                        pass

                elif "want to relax" in AlfredQueryOffline_assistant:
                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)

                    speech.AlfredSpeak(f'here is something to relax to,
hope you enjoy it, sir')

os.startfile(Alfred_config.DRIVE_LETTER+'Python_Env//New_Virtual_Env//Pro
ject_Files//My_Mp3//chill songs.mp3')
                    wait()

                    try :
                        if MyToDoListEdited != []:
                            MyToDoListEdited.pop(0)
                            AlfredQueryOfflineNew = MyToDoListEdited
                            print("")
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                            print('')
                    except:
                        pass

                    speech.AlfredSpeak(f'What else would you like me do
do, sir?')
                    print('')
                    print('listening ...')
                    print('')

                elif "the time" in AlfredQueryOffline_assistant:

                    print(f"\n FINAL AlfredQueryOffline for ASSISTANT
TIME CHECK : {AlfredQueryOffline} \n")

                    My_Message_Did = AlfredQueryOffline
                    NewPromptEdited_Did = My_Message_Did

                    # Apply replacements
```

```
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what's", "I told you")  # added contraction
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ', Alfred_No_Ollama)

                    assistant.timeOfDay(AlfredQueryOffline_assistant)

                elif ('the date' in AlfredQueryOffline_assistant):

                    AlfredQueryOffline = AlfredQueryOffline.replace("'",
"" )

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
```

```
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    My_Message_Doing = AlfredQueryOffline

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)

                    assistant.date(AlfredQueryOffline_assistant)

                elif ('the day' in AlfredQueryOffline_assistant):

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
```

```
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)

                    assistant.day(AlfredQueryOffline_assistant)

                elif ("good morning" in AlfredQueryOffline_assistant or
"good afternoon" in AlfredQueryOffline_assistant
                      or "good evening" in AlfredQueryOffline_assistant
or "good day" in AlfredQueryOffline_assistant):

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)

                    assistant.Geeting(AlfredQueryOffline_assistant)

                elif ("fuck" in AlfredQueryOffline_assistant or "mother
fucker" in AlfredQueryOffline_assistant
                      or "puss" in AlfredQueryOffline_assistant or "push"
in AlfredQueryOffline_assistant):

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline
```

```
                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)

                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)

                    print("you said : " + AlfredQueryOffline )

                    resSwearing =
open(Alfred_config.DRIVE_LETTER+"Python_Env\\New_Virtual_Env\\Personal\\S
wearing.txt")

                    try :
                        if MyToDoListEdited != []:
                            MyToDoListEdited.pop(0)
                            AlfredQueryOfflineNew = MyToDoListEdited
                            print("")
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                        ##    print(AlfredQueryOfflineNew)
                            print('')
                    except:
                        pass

                    resSwearing_Repeat =
open(Alfred_config.DRIVE_LETTER+"Python_Env\\New_Virtual_Env\\Personal\\S
wearing.txt")

                    print("")
```

```
                    print("About me...Repeat")
                    print("")

                    Alfred_Repeat_Previous_Response =
(resSwearing_Repeat.read())

                    response = Alfred_Repeat_Previous_Response
                    current_date = datetime.datetime.now().strftime('%Y-
%m-%d')
                    current_time =
datetime.datetime.now().strftime('%H:%M:%S')

                    chat_entry = {"date": current_date, "time":
current_time, "query": AlfredQueryOffline, "response": response}

                    memory.add_to_memory(chat_entry)  # □ Store in memory
                    repeat.add_to_repeat(chat_entry)  # □ Store last
response

                    print(f"Alfred_Repeat_Previous_Response:
{Alfred_Repeat_Previous_Response}")

                    print("")
                    print("About me...Repeat...Finished")
                    print("")

                    speech.AlfredSpeak(" sorry sir...?" +
resSwearing.read())

                    # GUI log if available
                    query_msg = (
                        f"At {current_date} :  {current_time} : You
Asked: {AlfredQueryOffline} "
                        f"and I
replied:\n\n{Alfred_Repeat_Previous_Response}\n\n"
                    )

                    model = "Alfred"
                    query_resp = f"At {current_date} :  {current_time} :
{model} : {Alfred_Repeat_Previous_Response} : {username}"

                    # Use the gui instance passed into main() to log;
handle None or exceptions gracefully
                    try:
                        if gui is not None:
                            try:
                                gui.log_message(query_msg)
                                gui.log_response(query_resp)
                            except Exception as e:
                                print("Error logging to GUI instance:",
e)
                        else:
                            print("GUI instance not available for logging
message (gui is None).")
```

```
                    except Exception as e:
                        print("Error while attempting GUI logging:", e)

                elif ('a fact' in AlfredQueryOffline_assistant or
'something interesting' in AlfredQueryOffline_assistant or 'interesting
fact' in AlfredQueryOffline_assistant
                    or 'another fact' in AlfredQueryOffline_assistant or
'awesome fact' in AlfredQueryOffline_assistant
                    or 'what can you tell me' in
AlfredQueryOffline_assistant or 'what can you tell me' in
AlfredQueryOffline_assistant
                    or 'what else can you tell me' in
AlfredQueryOffline_assistant):

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what else can you tell me", "I told you
something else")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)

                    try :
                        if MyToDoListEdited != []:
                            MyToDoListEdited.pop(0)
                            AlfredQueryOfflineNew = MyToDoListEdited
                            print("")
```

```
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                            print('')
                    except:
                        pass

                    assistant.fact(AlfredQueryOffline_assistant)

                elif ('cool facts' in AlfredQueryOffline_assistant or
'interesting facts' in AlfredQueryOffline_assistant or 'some facts' in
AlfredQueryOffline_assistant
                      or 'couple of facts' in
AlfredQueryOffline_assistant or 'awesome facts' in
AlfredQueryOffline_assistant):

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)

                    assistant.facts(AlfredQueryOffline_assistant)

                #jokes
                elif ('a joke' in AlfredQueryOffline_assistant or 'tell a
joke' in AlfredQueryOffline_assistant
```

```
                      or 'someting funny' in
AlfredQueryOffline_assistant):

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)

                    assistant.jokesAll(AlfredQueryOffline_assistant)

                #jokes
                elif ('tell me a twister joke' in
AlfredQueryOffline_assistant or 'twister joke' in
AlfredQueryOffline_assistant):

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
```

```
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)

                    assistant.jokesTwister(AlfredQueryOffline_assistant)

                #jokes
                elif ('tell me a tech joke' in
AlfredQueryOffline_assistant or 'tech joke' in
AlfredQueryOffline_assistant):

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")
```

```
                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)

                    assistant.jokesNeutral(AlfredQueryOffline_assistant)

                #jokes
                elif ('tell me a chuck' in AlfredQueryOffline_assistant
or 'chuck' in AlfredQueryOffline_assistant):

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)

                    assistant.jokesChuck(AlfredQueryOffline_assistant)

                elif ('some jokes' in AlfredQueryOffline_assistant or
'few jokes' in AlfredQueryOffline_assistant
```

```
                    or 'couple of jokes' in AlfredQueryOffline_assistant
or 'more jokes' in AlfredQueryOffline_assistant):

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)

assistant.JokesAllPlenty(AlfredQueryOffline_assistant)

                elif 'want coffee' in AlfredQueryOffline_assistant or
'some coffee' in AlfredQueryOffline_assistant or 'coffee plaese' in
AlfredQueryOffline_assistant:

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
```

```
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)
                    print('sir: ' + AlfredQueryOffline)

                    try :
                        if MyToDoListEdited != []:
                            MyToDoListEdited.pop(0)
                            AlfredQueryOfflineNew = MyToDoListEdited
                            print("")
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                        ##    print(AlfredQueryOfflineNew)
                            print('')
                    except:
                        pass

                    print("")
                    print("About me...Repeat")
                    print("")

                    Alfred_Repeat_Previous_Response = "due to unforseen
difficulties. I am not able to do that, Sir. Go and make your own damn
coffee..?"
                    print(f"Alfred_Repeat_Previous_Response:
{Alfred_Repeat_Previous_Response}")

                    response = Alfred_Repeat_Previous_Response
                    current_date = datetime.datetime.now().strftime('%Y-
%m-%d')
                    current_time =
datetime.datetime.now().strftime('%H:%M:%S')

                    chat_entry = {"date": current_date, "time":
current_time, "query": AlfredQueryOffline, "response": response}
```

```
                    memory.add_to_memory(chat_entry)  # □ Store in memory
                    repeat.add_to_repeat(chat_entry)  # □ Store last
response

                    print("")
                    print("About me...Repeat...Finished")
                    print("")

                    listen.send_bluetooth("due to unforseen difficulties.
I am not able to do that, Sir. Go and make your own damn coffee..?")
                    speech.AlfredSpeak("due to unforseen difficulties. I
am not able to do that, Sir. Go and make your own damn coffee..?")


                    # GUI log if available
                    query_msg = (
                        f"At {current_date} :  {current_time} : You
Asked: {AlfredQueryOffline} "
                        f"and I
replied:\n\n{Alfred_Repeat_Previous_Response}\n\n"
                    )

                    model = "Alfred"
                    query_resp = f"At {current_date} :  {current_time} :
{model} : {Alfred_Repeat_Previous_Response} : {username}"


                    # Use the gui instance passed into main() to log;
handle None or exceptions gracefully
                    try:
                        if gui is not None:
                            try:
                                gui.log_message(query_msg)
                                gui.log_response(query_resp)
                            except Exception as e:
                                print("Error logging to GUI instance:",
e)
                        else:
                            print("GUI instance not available for logging
message (gui is None).")
                    except Exception as e:
                        print("Error while attempting GUI logging:", e)

                    listen.send_bluetooth("Listening...")
                    speech.AlfredSpeak("Listening...")


            ##            #Alfred features

                elif ('your friend' in AlfredQueryOffline_assistant or
'your buddy' in AlfredQueryOffline_assistant
                    or 'your chum' in AlfredQueryOffline_assistant or
'your compadre' in AlfredQueryOffline_assistant):
```

```
                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)
                    resYourFriend =
open(Alfred_config.DRIVE_LETTER+"Python_Env\\New_Virtual_Env\\Personal\\A
lfred_friends.txt")

                    try :
                        if MyToDoListEdited != []:
                            MyToDoListEdited.pop(0)
                            AlfredQueryOfflineNew = MyToDoListEdited
                            print("")
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                        ##    print(AlfredQueryOfflineNew)
                            print('')
                    except:
                        pass

                    resYourFriend_Repeat =
open(Alfred_config.DRIVE_LETTER+"Python_Env\\New_Virtual_Env\\Personal\\A
lfred_friends.txt")
```

```
                    print("")
                    print("About me...Repeat")
                    print("")

                    Alfred_Repeat_Previous_Response =
(resYourFriend_Repeat.read())
                    print(f"Alfred_Repeat_Previous_Response:
{Alfred_Repeat_Previous_Response}")

                    print("")
                    print("About me...Repeat...Finished")
                    print("")

                    response = Alfred_Repeat_Previous_Response
                    current_date = datetime.datetime.now().strftime('%Y-
%m-%d')
                    current_time =
datetime.datetime.now().strftime('%H:%M:%S')

                    chat_entry = {"date": current_date, "time":
current_time, "query": AlfredQueryOffline, "response": response}

                    memory.add_to_memory(chat_entry)  # □ Store in memory
                    repeat.add_to_repeat(chat_entry)  # □ Store last
response

                    My_Friend_Response = resYourFriend.read()

                    listen.send_bluetooth(f" o k, here we go...?
{My_Friend_Response}")
                    speech.AlfredSpeak(f" o k, here we go...?
{My_Friend_Response}")


                    # GUI log if available
                    query_msg = (
                        f"At {current_date} :  {current_time} : You
Asked: {AlfredQueryOffline} "
                        f"and I
replied:\n\n{Alfred_Repeat_Previous_Response}\n\n"
                    )

                    model = "Alfred"
                    query_resp = f"At {current_date} :  {current_time} :
{model} : {Alfred_Repeat_Previous_Response} : {username}"

                    # Use the gui instance passed into main() to log;
handle None or exceptions gracefully
                    try:
                        if gui is not None:
                            try:
                                gui.log_message(query_msg)
                                gui.log_response(query_resp)
                            except Exception as e:
```

```
                                print("Error logging to GUI instance:",
e)
                        else:
                            print("GUI instance not available for logging
message (gui is None).")
                    except Exception as e:
                        print("Error while attempting GUI logging:", e)

                    listen.send_bluetooth("Listening...")
                    speech.AlfredSpeak("Listening...")

                #Alfred features

                elif ('his friend matt' in AlfredQueryOffline_assistant
or 'his buddy matt' in AlfredQueryOffline_assistant
                    or 'his chum matt' in AlfredQueryOffline_assistant or
'his compadre matt' in AlfredQueryOffline_assistant
                    or 'buddy matt' in AlfredQueryOffline_assistant):

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)
```

```
                    resSebasFriend =
open(Alfred_config.DRIVE_LETTER+"Python_Env\\New_Virtual_Env\\Personal\\M
at_Sebastiaan.txt")

                    try :
                        if MyToDoListEdited != []:
                            MyToDoListEdited.pop(0)
                            AlfredQueryOfflineNew = MyToDoListEdited
                            print("")
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                        ##    print(AlfredQueryOfflineNew)
                            print('')
                    except:
                        pass

                    resSebasFriend_Repeat =
open(Alfred_config.DRIVE_LETTER+"Python_Env\\New_Virtual_Env\\Personal\\M
at_Sebastiaan.txt")

                    print("")
                    print("About me...Repeat")
                    print("")

                    Alfred_Repeat_Previous_Response =
(resSebasFriend_Repeat.read())
                    print(f"Alfred_Repeat_Previous_Response:
{Alfred_Repeat_Previous_Response}")

                    response = Alfred_Repeat_Previous_Response
                    current_date = datetime.datetime.now().strftime('%Y-
%m-%d')
                    current_time =
datetime.datetime.now().strftime('%H:%M:%S')

                    chat_entry = {"date": current_date, "time":
current_time, "query": AlfredQueryOffline, "response": response}

                    memory.add_to_memory(chat_entry)  # □ Store in memory
                    repeat.add_to_repeat(chat_entry)  # □ Store last
response

                    print("")
                    print("About me...Repeat...Finished")
                    print("")

                    listen.send_bluetooth(" o k, here we go...?" +
resSebasFriend.read())
                    speech.AlfredSpeak(" o k, here we go...?"  +
resSebasFriend.read())

                    # GUI log if available
                    query_msg = (
```

```
                        f"At {current_date} :  {current_time} : You
Asked: {AlfredQueryOffline} "
                        f"and I
replied:\n\n{Alfred_Repeat_Previous_Response}\n\n"
                    )

                    model = "Alfred"
                    query_resp = f"At {current_date} :  {current_time} :
{model} : {Alfred_Repeat_Previous_Response} : {username}"

                    # Use the gui instance passed into main() to log;
handle None or exceptions gracefully
                    try:
                        if gui is not None:
                            try:
                                gui.log_message(query_msg)
                                gui.log_response(query_resp)
                            except Exception as e:
                                print("Error logging to GUI instance:",
e)
                        else:
                            print("GUI instance not available for logging
message (gui is None).")
                    except Exception as e:
                        print("Error while attempting GUI logging:", e)

                    listen.send_bluetooth("Listening...")
                    speech.AlfredSpeak("Listening...")

            ##      System Power Alfred_config

                elif ('logout of windows' in
AlfredQueryOffline_assistant):

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
```

```
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)

                    try :
                        if MyToDoListEdited != []:
                            MyToDoListEdited.pop(0)
                            AlfredQueryOfflineNew = MyToDoListEdited
                            print("")
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                            print('')
                    except:
                        pass

                    os.system("shutdown -1")

                elif ('restart computer' in AlfredQueryOffline_assistant
or 'restart pc' in AlfredQueryOffline_assistant
                        or 'reboot pc' in AlfredQueryOffline_assistant or
'reboot computer' in AlfredQueryOffline_assistant
                        or 'reboot system' in
AlfredQueryOffline_assistant):

                    My_Message_Doing = AlfredQueryOffline

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)

                    try :
                        if MyToDoListEdited != []:
                            MyToDoListEdited.pop(0)
                            AlfredQueryOfflineNew = MyToDoListEdited
                            print("")
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                            print('')
                    except:
                        pass

                    os.system("shutdown /r /t 1")
```

```
                elif ('shut down computer' in
AlfredQueryOffline_assistant or 'shut down pc' in
AlfredQueryOffline_assistant):

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)

                    try :
                        if MyToDoListEdited != []:
                            MyToDoListEdited.pop(0)
                            AlfredQueryOfflineNew = MyToDoListEdited
                            print("")
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                        ##    print(AlfredQueryOfflineNew)
                            print('')
                    except:
                        pass

                    os.system("shutdown /r /t 1")

                #play songs
```

```
                elif ('play some music' in AlfredQueryOffline_assistant
or 'play another song' in AlfredQueryOffline_assistant or 'play another
mp3' in AlfredQueryOffline_assistant
                    or 'another tune' in AlfredQueryOffline_assistant or
'next mp3' in AlfredQueryOffline_assistant or 'next tune' in
AlfredQueryOffline_assistant
                    or 'lots of music' in AlfredQueryOffline_assistant or
'plenty of music' in AlfredQueryOffline_assistant or 'skip tune' in
AlfredQueryOffline_assistant
                    or 'many songs' in AlfredQueryOffline_assistant or
'skip song' in AlfredQueryOffline_assistant or 'play some tunes' in
AlfredQueryOffline_assistant
                    or 'skip mp3' in AlfredQueryOffline_assistant or
'songs' in AlfredQueryOffline_assistant  or 'play tunes' in
AlfredQueryOffline_assistant
                    or 'play all songs' in AlfredQueryOffline_assistant
or 'some tunes' in AlfredQueryOffline_assistant
                    or 'skip this song' in AlfredQueryOffline_assistant
or 'skip the song' in AlfredQueryOffline_assistant
                    or 'skip that song' in AlfredQueryOffline_assistant
or 'play me some music' in AlfredQueryOffline_assistant):

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
```

```
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)
                    speech.AlfredSpeak("Playing....")

                    files =
Alfred_config.DRIVE_LETTER+"Python_Env//New_Virtual_Env//Project_Files//M
y_Mp3//Playlists//All//All.m3u8"
                    dir =
Alfred_config.DRIVE_LETTER+"Python_Env//New_Virtual_Env//Project_Files//M
y_Mp3//Playlists//All"
                    filename = random.choice(os.listdir(dir))
                    path = os.path.join(dir, filename)
                    os.startfile(os.path.join(dir, filename))
                    print ("my new mp3 list : ")
                    print ((dir)+ (filename))

                    try :
                        if MyToDoListEdited != []:
                            MyToDoListEdited.pop(0)
                            AlfredQueryOfflineNew = MyToDoListEdited
                            print("")
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                            print('')
                    except:
                        pass

                elif ('play disc' in AlfredQueryOffline_assistant or
'skip disc' in AlfredQueryOffline_assistant or 'play another disc mp3' in
AlfredQueryOffline_assistant
                    or 'another disc tune' in
AlfredQueryOffline_assistant or 'next disc mp3' in
AlfredQueryOffline_assistant or 'next disc tune' in
AlfredQueryOffline_assistant
                    or 'play next disc' in AlfredQueryOffline_assistant
or 'skip mp3 disc' in AlfredQueryOffline_assistant or 'skip disc tune '
in AlfredQueryOffline_assistant
                    or 'skip disc music' in AlfredQueryOffline_assistant
or 'play some disc tunes' in AlfredQueryOffline_assistant
                    or 'skip disc tune' in AlfredQueryOffline_assistant
or 'play disc tunes' in AlfredQueryOffline_assistant or 'play disc tunes'
in AlfredQueryOffline_assistant):

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
```

```
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)
                    speech.AlfredSpeak("Playing CD....")

                    files = ("d:\\*.mp3" or "d:\\*.cda")
                    dir = "D:\\"
                    filename = random.choice(os.listdir(dir))
                    path = os.path.join(dir, filename)
                    os.startfile(os.path.join(dir, filename))
                    print ("my new cd list : ")
                    print ((dir)+ (filename))

                    try :
                        if MyToDoListEdited != []:
                            MyToDoListEdited.pop(0)
                            AlfredQueryOfflineNew = MyToDoListEdited
                            print("")
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                            print('')
                    except:
                        pass

                elif ('play my song' in AlfredQueryOffline_assistant or
'my mp3' in AlfredQueryOffline_assistant or 'my tune' in
AlfredQueryOffline_assistant
                      or 'favorite song' in AlfredQueryOffline_assistant
or 'favorite tune' in AlfredQueryOffline_assistant):

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did
```

```
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)

                    try :
                        if MyToDoListEdited != []:
                            MyToDoListEdited.pop(0)
                            AlfredQueryOfflineNew = MyToDoListEdited
                            print("")
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                            print('')
                    except:
                        pass

                    speech.AlfredSpeak("Playing your song")

                    files =
Alfred_config.DRIVE_LETTER+"Python_Env//New_Virtual_Env//Project_Files//M
y_Mp3//Playlists//Favourite//Favourite.m3u8"
                    dir =
Alfred_config.DRIVE_LETTER+"Python_Env//New_Virtual_Env//Project_Files//M
y_Mp3//Playlists//Favourite"

                    filename = random.choice(os.listdir(dir))
                    path = os.path.join(dir, filename)
                    os.startfile(os.path.join(dir, filename))
                    print ("my new favourite mp3 list : ")
                    print ((dir)+ (filename))
```

```
                    try :
                        if MyToDoListEdited != []:
                            MyToDoListEdited.pop(0)
                            AlfredQueryOfflineNew = MyToDoListEdited
                            print("")
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                            print('')
                    except:
                        pass

                #screenshot
                elif ('screenshot' in AlfredQueryOffline):

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)
                    assistant.screenshot(AlfredQueryOffline_assistant)
                    speech.AlfredSpeak("Done!")

                #cpu and battery usage
                elif ('cpu and battery' in AlfredQueryOffline or 'Laptop
battery' in AlfredQueryOffline or 'the battery status' in
AlfredQueryOffline
```

```
                    or 'cpu' in AlfredQueryOffline):

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)

                    assistant.cpu(AlfredQueryOffline_assistant)

                    #Alfred read pdf

            ##                elif ('read bible chapter' in
AlfredQueryOffline or 'read pdf bible chapter' in AlfredQueryOffline
            ##                    or 'bible chapter' in
AlfredQueryOffline or 'a bible chapter' in AlfredQueryOffline
            ##                    or 'PDF bible chapter' in
AlfredQueryOffline  or 'the bible chapter' in AlfredQueryOffline):
            ##
            ##                    print('sir: ' + AlfredQueryOffline)
            ##                    Alfred_No_Ollama = 0
            ##                    print('Alfred_No_Ollama : ' ,
Alfred_No_Ollama)
            ##                    pdf_Bible_Chapter_reader()
            ##
            ##
```

```
            ##                      #Alfred read pdf
            ##
            ##                  elif ('read a bible page' in
AlfredQueryOffline or 'read pdf bible page' in AlfredQueryOffline
            ##                      or 'bible page' in AlfredQueryOffline
or 'a bible page' in AlfredQueryOffline
            ##                      or 'a bible' in AlfredQueryOffline  or
'the bible' in AlfredQueryOffline):
            ##
            ##                      print('sir: ' + AlfredQueryOffline)
            ##                      Alfred_No_Ollama = 0
            ##                      print('Alfred_No_Ollama : ' ,
Alfred_No_Ollama)
            ##                      pdf_Bible_Page_reader()


##////////////////////////////////////////////////////////////////////////
///////////

##////////////////////////////////////////////////////////////////////////
///////////

                elif 'watch anunnaki movie' in AlfredQueryOffline or
'play anunnaki movie' in AlfredQueryOffline:

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")
```

```
                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)
                    speech.AlfredSpeak("Playing anunaki movie....")

                    try :
                        if MyToDoListEdited != []:
                            MyToDoListEdited.pop(0)
                            AlfredQueryOfflineNew = MyToDoListEdited
                            print("")
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                            print('')
                    except:
                        pass


startfile(Alfred_config.DRIVE_LETTER+"Python_Env\\New_Virtual_Env\\Projec
t_Files\\My_MovieZ\\Lost Books\\ANUNNAKI MOVIE _ ANUNNAKI FULL MOVIE 2024
_ Lost book of Enki Complete Story.mp4")


                elif 'watch enoch movie' in AlfredQueryOffline or 'play
enoch movie' in AlfredQueryOffline:

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
```

```
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)
                    speech.AlfredSpeak("Playing enoch movie....")
                    listen.send_bluetooth("Playing enoch movie....")

                    try :
                        if MyToDoListEdited != []:
                            MyToDoListEdited.pop(0)
                            AlfredQueryOfflineNew = MyToDoListEdited
                            print("")
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                            print('')
                    except:
                        pass


startfile(Alfred_config.DRIVE_LETTER+"Python_Env\\New_Virtual_Env\\Projec
t_Files\\My_MovieZ\\Lost Books\\The Book of Enoch banned from the Bible
reveals shocking mysteries of our history!.mp4")

                elif ('about the crops' in AlfredQueryOffline):

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
```

```
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)

                    assistant.crops(AlfredQueryOffline_assistant)

                elif ('stop alfred' in AlfredQueryOffline or 'finished
alfred' in AlfredQueryOffline  or 'end alfred' in AlfredQueryOffline
                    or 'stop running' in AlfredQueryOffline or 'quit
alfred' in AlfredQueryOffline or "quit alfred" in AlfredQueryOffline
                    or "cheers" in AlfredQueryOffline or "good bye" in
AlfredQueryOffline or "bye bye" in AlfredQueryOffline
                      or "stop running software" in AlfredQueryOffline or
"goodbye" in AlfredQueryOffline or "alfred stop" in AlfredQueryOffline):

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)

                    listen.send_bluetooth("It was a pleasure working with
you")
                    listen.send_bluetooth("Thank you for using me Sir")
```

```
                    listen.send_bluetooth("Alfred session stopped. See
you later...?")
                    listen.send_bluetooth("Good bye, sir")

                    speech.AlfredSpeak("It was a pleasure working with
you")
                    speech.AlfredSpeak("Thank you for using me Sir")
                    speech.AlfredSpeak("Alfred session stopped. See you
later...?")
                    speech.AlfredSpeak("Good bye, sir")

                    print("Alfred session stopped. See you later...?")

                    engine.runAndWait()
                    engine.stop()


playsound(Alfred_config.DRIVE_LETTER+'Python_Env//New_Virtual_Env//Projec
t_Files//My_Mp3//Windows Notify Calendar.wav')
                    cls

                    try :
                        if MyToDoListEdited != []:
                            MyToDoListEdited.pop(0)
                            AlfredQueryOfflineNew = MyToDoListEdited
                            print("")
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                            print('')
                    except:
                        pass

                    time.sleep(5)
                    sys.exit(0)

                elif ('at the door' in AlfredQueryOffline or 'at the
door' in AlfredQueryOffline or 'at the door' in AlfredQueryOffline
                    or 'is the door' in AlfredQueryOffline):

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
```

```
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)

                    listen.send_bluetooth("excuse me who is at the door,
listening")
                    speech.AlfredSpeak ("excuse me who is at the door,
listening")

                    vosk_model = setup_vosk_model()
                    text_to_speech_engine = setup_text_to_speech()

                    try :
                        if MyToDoListEdited != []:
                            MyToDoListEdited.pop(0)
                            AlfredQueryOfflineNew = MyToDoListEdited
                            print("")
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                            print('')
                    except:
                        pass

                    while True:

                        audio_input = listen_to_audio_Door()
                        recognized_text = recognize_speech(vosk_model,
audio_input)
                        print("You said:", recognized_text)

                        VisitorName = recognized_text

                        if VisitorName is not None:
                            speech.AlfredSpeak("It is")
                            speech.AlfredSpeak(VisitorName)
                            speech.AlfredSpeak("What would you like me to
do, sir ...?")
                            listen.send_bluetooth(f"It is {VisitorName} .
What would you like me to do, sir...?")
```

```
                            return main(assistant, gui)

                        else:

                            audio_input = listen_to_audio_Door()
                            recognized_text =
recognize_speech(vosk_model, audio_input)

                elif ("don't want to see" in AlfredQueryOffline or "can't
see" in AlfredQueryOffline or 'i am busy' in AlfredQueryOffline
                    or 'tell them to go' in AlfredQueryOffline):

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)

                    print('sir: ' + AlfredQueryOffline)

                    listen.send_bluetooth("I am very sorry, sir. My
master is currently not here. He cant see you now. Please come back
later. Thank you very much. Have a nice day..? ")
                    speech.AlfredSpeak("I am very sorry, sir. My master
is currently not here. He cant see you now. Please come back later. Thank
you very much. Have a nice day..? ")
```

```
                    try :
                        if MyToDoListEdited != []:
                            MyToDoListEdited.pop(0)
                            AlfredQueryOfflineNew = MyToDoListEdited
                            print("")
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                            print('')
                    except:
                        pass

                    return main(assistant, gui)

                elif ('let them in ' in AlfredQueryOffline or 'let him in
' in AlfredQueryOffline or 'let her in ' in AlfredQueryOffline
                    or ' bring them ' in AlfredQueryOffline or ' open for
' in AlfredQueryOffline):

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    Alfred_No_Ollama = 0

                    print('sir: ' + AlfredQueryOffline)

                    try :
                        if MyToDoListEdited != []:
```

```
                            MyToDoListEdited.pop(0)
                            AlfredQueryOfflineNew = MyToDoListEdited
                            print("")
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                            print('')
                    except:
                        pass


home_auto.Arduino_Home_Automation_Send_Door_Open2(AlfredQueryOffline)


##////////////////////////////////////////////////////////////////////
//////////

                elif ("wait a minute" in AlfredQueryOffline or "wait a
bit" in AlfredQueryOffline
                    or "hold on" in AlfredQueryOffline):

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)
```

```
                    try :
                        if MyToDoListEdited != []:
                            MyToDoListEdited.pop(0)
                            AlfredQueryOfflineNew = MyToDoListEdited
                            print("")
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                            print('')
                    except:
                        pass


assistant.AlfredStopListeningTimer(AlfredQueryOffline_assistant)

                elif ("clear counter" in AlfredQueryOffline or "the
counter" in AlfredQueryOffline):

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)


                    try :
                        if MyToDoListEdited != []:
```

```
                            MyToDoListEdited.pop(0)
                            AlfredQueryOfflineNew = MyToDoListEdited
                            print("")
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                            print('')
                    except:
                        pass

                if any(trigger in AlfredQueryOffline for trigger in
_pause_triggers):
                    # keep original messages for logs
                    My_Message_Doing = AlfredQueryOffline
                    My_Message_Did = AlfredQueryOffline

                    # Build edited prompt (do replacements on a lower-
cased copy to be robust)
                    NewPromptEdited_Did = My_Message_Did.lower()

                    # Replacements (mirrors your original replacements,
done case-insensitively)
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "i told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "i told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "i told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "i told you who i am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "i told you what i am")

                    # Optional: tidy whitespace
                    NewPromptEdited_Did = "
".join(NewPromptEdited_Did.split())

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    # Update Did_History (preserve your original
insertion order)
                    try:
                        Did_History.insert(0, f"{NewPromptEdited_Did}")
                        Did_History.insert(1, f". and . ")
                    except Exception as e:
                        # If Did_History isn't a list or is missing,
print error but continue
```

```
                        print("Warning: Could not update Did_History:",
e)

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ', Alfred_No_Ollama)

                    # Notify user (voice + text)
                    speech.AlfredSpeak("Paused, sir. Please press the
SPACE BAR to continue.")
                    print("Paused, sir. Please press the SPACE BAR to
continue.")

                    # small pause so speak output starts
                    time.sleep(0.25)

                    # Try to pop the next todo if present (same logic as
your original try block)
                    try:
                        if isinstance(MyToDoListEdited, list) and
MyToDoListEdited:
                            MyToDoListEdited.pop(0)
                            AlfredQueryOfflineNew = MyToDoListEdited
                            print("")
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                            print("")
                    except Exception:
                        pass

                    # Wait for the user to press SPACE (no pre-consumed
read_key)
                    try:
                        while True:
                            key = keyboard.read_key()  # blocks until a
key is pressed
                            if key in ("space", " "):
                                # Clear screen (optional)
                                os.system('cls' if os.name == 'nt' else
'clear')
                                # Wake up assistant (use the assistant
variable you already have)
                                try:

assistant.Wake_up_Greeting(AlfredQueryOffline_assistant)
                                except Exception as e:
                                    print("Warning:
assistant.Wake_up_Greeting failed:", e)
                                break
                            # you could also accept any key to continue:
                            # if key: break
                    except KeyboardInterrupt:
                        # allow Ctrl+C to break out cleanly if needed
                        print("\nInterrupted while waiting for SPACE.")
```

```
                    except Exception as e:
                        print("Error while waiting for keypress:", e)

                elif ("restart system" in AlfredQueryOffline or "reset
system" in AlfredQueryOffline
                      or "restart the system" in AlfredQueryOffline or
"reset the system" in AlfredQueryOffline):

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)
                    speech.AlfredSpeak("Ok sir, Restarting the program")
                    cls

                    try :
                        if MyToDoListEdited != []:
                            MyToDoListEdited.pop(0)
                            AlfredQueryOfflineNew = MyToDoListEdited
                            print("")
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                            print('')
                    except:
                        pass
```

```
                    os.execl(sys.executable, os.path.abspath(__file__),
*sys.argv)

                #personal info
                elif ('something about you' in AlfredQueryOffline or
'about yourself' in AlfredQueryOffline
                    or 'who are you' in AlfredQueryOffline or 'about
alfred' in AlfredQueryOffline
                    or 'about your self' in AlfredQueryOffline  or 'about
your own' in AlfredQueryOffline):

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something about you", "something about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", " you about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about yourself", " you about my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "who am I")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)

                    assistant.personal(AlfredQueryOffline_assistant)

                elif ('old are you' in AlfredQueryOffline or 'your age'
in AlfredQueryOffline or 'when were you born' in AlfredQueryOffline):

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline
```

```
                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)
                    assistant.YourAge(AlfredQueryOffline_assistant)

                elif ('your developer' in AlfredQueryOffline or 'your
developer' in AlfredQueryOffline
                      or 'your father' in AlfredQueryOffline or 'who
developed you' in AlfredQueryOffline
                      or 'your creator' in AlfredQueryOffline or 'who
created you' in AlfredQueryOffline
                      or 'your daddy' in AlfredQueryOffline):

                    print(f"\n FINAL AlfredQueryOffline for ASSISTANT
DEVELOPER BEFORE CHECK : {AlfredQueryOffline} \n")

                    My_Message_Doing = AlfredQueryOffline
                    My_Message_Doing = My_Message_Doing.replace("tell me
about your", "I am telling you about my")

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
```

```
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about your", "I told you about my")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)
                    resCreator =
open(Alfred_config.DRIVE_LETTER+"Python_Env\\New_Virtual_Env\\Personal\\a
bout.txt")


                    try :
                        if MyToDoListEdited != []:
                            MyToDoListEdited.pop(0)
                            AlfredQueryOfflineNew = MyToDoListEdited
                            print("")
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                            print('')
                    except:
                        pass

                    resCreator_Repeat =
open(Alfred_config.DRIVE_LETTER+"Python_Env\\New_Virtual_Env\\Personal\\a
bout.txt")

                    print("")
                    print("About me...Repeat")
                    print("")

                    Alfred_Repeat_Previous_Response =
(resCreator_Repeat.read())
                    print(f"Alfred_Repeat_Previous_Response:
{Alfred_Repeat_Previous_Response}")

                    print("")
                    print("About me...Repeat...Finished")
```

```
                    print("")

                    Developer_Read = resCreator.read()

                    response = Alfred_Repeat_Previous_Response
                    current_date = datetime.datetime.now().strftime('%Y-
%m-%d')
                    current_time =
datetime.datetime.now().strftime('%H:%M:%S')

                    chat_entry = {"date": current_date, "time":
current_time, "query": AlfredQueryOffline, "response": response}

                    memory.add_to_memory(chat_entry)  # □ Store in memory
                    repeat.add_to_repeat(chat_entry)  # □ Store last
response

                    # GUI log if available
                    query_msg = (
                        f"At {current_date} :  {current_time} : You
Asked: {AlfredQueryOffline} "
                        f"and I
replied:\n\n{Alfred_Repeat_Previous_Response}\n\n"
                    )

                    model = "Alfred"
                    query_resp = f"At {current_date} :  {current_time} :
{model} : {Alfred_Repeat_Previous_Response} : {username}"

                    # Use the gui instance passed into main() to log;
handle None or exceptions gracefully
                    try:
                        if gui is not None:
                            try:
                                gui.log_message(query_msg)
                                gui.log_response(query_resp)
                            except Exception as e:
                                print("Error logging to GUI instance:",
e)
                        else:
                            print("GUI instance not available for logging
message (gui is None).")
                    except Exception as e:
                        print("Error while attempting GUI logging:", e)

                    listen.send_bluetooth(response)
                    speech.AlfredSpeak(response)

                elif ('what else can you do' in AlfredQueryOffline or
'about your functions' in AlfredQueryOffline
                    or 'about your features' in AlfredQueryOffline or
'what can you' in AlfredQueryOffline):

                    My_Message_Doing = AlfredQueryOffline
```

```
                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)

                    print('sir: ' + AlfredQueryOffline)
                    resAbout =
open(Alfred_config.DRIVE_LETTER+"Python_Env\\New_Virtual_Env\\Personal\\F
eatures.txt")

                    try :
                        if MyToDoListEdited != []:
                            MyToDoListEdited.pop(0)
                            AlfredQueryOfflineNew = MyToDoListEdited
                            print("")
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                            print('')
                    except:
                        pass

                    resAbout_Repeat =
open(Alfred_config.DRIVE_LETTER+"Python_Env\\New_Virtual_Env\\Personal\\F
eatures.txt")

                    print("")
                    print("About me...Repeat")
```

```
                    print("")

                    Alfred_Repeat_Previous_Response =
(resAbout_Repeat.read())
                    print(f"Alfred_Repeat_Previous_Response:
{Alfred_Repeat_Previous_Response}")

                    response = Alfred_Repeat_Previous_Response
                    current_date = datetime.datetime.now().strftime('%Y-
%m-%d')
                    current_time =
datetime.datetime.now().strftime('%H:%M:%S')

                    chat_entry = {"date": current_date, "time":
current_time, "query": AlfredQueryOffline, "response": response}

                    memory.add_to_memory(chat_entry)  # □ Store in memory
                    repeat.add_to_repeat(chat_entry)  # □ Store last
response

                    print("")
                    print("About me...Repeat...Finished")
                    print("")

                    # GUI log if available
                    query_msg = (
                        f"At {current_date} :  {current_time} : You
Asked: {AlfredQueryOffline} "
                        f"and I
replied:\n\n{Alfred_Repeat_Previous_Response}\n\n"
                    )

                    model = "Alfred"
                    query_resp = f"At {current_date} :  {current_time} :
{model} : {Alfred_Repeat_Previous_Response} : {username}"

                    # Use the gui instance passed into main() to log;
handle None or exceptions gracefully
                    try:
                        if gui is not None:
                            try:
                                gui.log_message(query_msg)
                                gui.log_response(query_resp)
                            except Exception as e:
                                print("Error logging to GUI instance:",
e)
                        else:
                            print("GUI instance not available for logging
message (gui is None).")
                    except Exception as e:
                        print("Error while attempting GUI logging:", e)


                    speech.AlfredSpeak(resAbout.read())
```

```
                    listen.send_bluetooth(resAbout.read())

                elif ("hello alfred" in AlfredQueryOffline or "hi there"
in AlfredQueryOffline  or "hi alfred" in AlfredQueryOffline
                      or "hey alfred" in AlfredQueryOffline or "whazz up"
in AlfredQueryOffline):

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)

                    print('sir: ' + AlfredQueryOffline)
                    responses = ["Hello! How can I assist you today?",
                                 "Hi there! What can I do for you?",
                                 "Hey! How's your day going?",
                                 "whazz up! Ready to start the day?",
                                 "Greetings! What brings you here?",
                                  "Hi! Ready for some assistance?"]

                    try :
                        if MyToDoListEdited != []:
                            MyToDoListEdited.pop(0)
                            AlfredQueryOfflineNew = MyToDoListEdited
                            print("")
```

```
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                            print('')
                    except:
                        pass

                    Greeting_Random = random.choice(responses)

                    print("")
                    print("About me...Repeat")
                    print("")

                    Alfred_Repeat_Previous_Response = (Greeting_Random)
                    print(f"Alfred_Repeat_Previous_Response:
{Alfred_Repeat_Previous_Response}")

                    response = Alfred_Repeat_Previous_Response
                    current_date = datetime.datetime.now().strftime('%Y-
%m-%d')
                    current_time =
datetime.datetime.now().strftime('%H:%M:%S')

                    chat_entry = {"date": current_date, "time":
current_time, "query": AlfredQueryOffline, "response": response}

                    memory.add_to_memory(chat_entry)  # □ Store in memory
                    repeat.add_to_repeat(chat_entry)  # □ Store last
response

                    print("")
                    print("About me...Repeat...Finished")
                    print("")

                    # GUI log if available
                    query_msg = (
                        f"At {current_date} :  {current_time} : You
Asked: {AlfredQueryOffline} "
                        f"and I
replied:\n\n{Alfred_Repeat_Previous_Response}\n\n"
                    )

                    model = "Alfred"
                    query_resp = f"At {current_date} :  {current_time} :
{model} : {Alfred_Repeat_Previous_Response} : {username}"

                    # Use the gui instance passed into main() to log;
handle None or exceptions gracefully
                    try:
                        if gui is not None:
                            try:
                                gui.log_message(query_msg)
                                gui.log_response(query_resp)
                            except Exception as e:
```

```
                                print("Error logging to GUI instance:",
e)
                        else:
                            print("GUI instance not available for logging
message (gui is None).")
                    except Exception as e:
                        print("Error while attempting GUI logging:", e)

                    listen.send_bluetooth(Greeting_Random)
                    speech.AlfredSpeak(Greeting_Random)

                elif ("gary" in AlfredQueryOffline or "clean" in
AlfredQueryOffline
                    or "pickup" in AlfredQueryOffline or "take away" in
AlfredQueryOffline
                    or "get that" in AlfredQueryOffline or "fetch" in
AlfredQueryOffline
                    or "pick up" in AlfredQueryOffline):

                    print('\n')
                    print('\r')
                    print("AlfredQueryOffline : " , AlfredQueryOffline)

                    My_Message_Doing = AlfredQueryOffline


                    My_Message_Did = AlfredQueryOffline
                    print('\n')
                    print('\r')
                    print("My_Message_Did : " , My_Message_Did)


                    NewPromptEdited = AlfredQueryOffline
                    print('\n')
                    print('\r')
                    print("NewPromptEdited : " , NewPromptEdited)

                    NewPromptEditedSpeak = NewPromptEdited
                    print('\n')
                    print('\r')
                    print("NewPromptEditedSpeak before : " ,
NewPromptEditedSpeak)

                    NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("something", "")
                    NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("some", "")
                    NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("in short", "")
                    NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("storie", "")
                    NewPromptEditedSpeak = NewPromptEditedSpeak.replace("
her ", "")
```

```
                    NewPromptEditedSpeak = NewPromptEditedSpeak.replace("
him ", "")
                    NewPromptEditedSpeak = NewPromptEditedSpeak.replace("
them ", "")
                    NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("how they", "how it")
                    NewPromptEditedSpeak = NewPromptEditedSpeak.replace("
they ", "")
                    NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("tell me what", "what")
                    NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("tell me about", "")
                    NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("tell me who", "who")
                    NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("write me", "writing you")
                    NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("do you", "do I")
                    NewPromptEditedSpeak = NewPromptEditedSpeak.replace("
then ", "")
                    NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("detail in front of you", "detail in front
of me")
                    NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("yourself", "my self")
                    NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("about your", "about my")
                    NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("is your", "is my")
                    NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("would your", "would my")
                    NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("is yours", "is mine")
                    NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("would you", "would I")
                    NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("what can you", "what can I")
                    NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("who can you", "who can I")
                    NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("can you", "")
                    NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("could you", "could I")
                    NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("tell me", "telling you")
                    NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("to me", "to you")
                    NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("create", "creating")
                    NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("me more", "")
                    NewPromptEditedSpeak = NewPromptEditedSpeak.replace("
me ", " you ")
```

```
                    NewPromptEditedSpeak = NewPromptEditedSpeak.replace("
do you ", " do I ")
                    NewPromptEditedSpeak = NewPromptEditedSpeak.replace("
will you ", " will I ")

                    print('\n')
                    print('\r')
                    print("NewPromptEditedSpeak after : " ,
NewPromptEditedSpeak)



                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)

                    print('sir: ' + AlfredQueryOffline)

                    try :
                        if MyToDoListEdited != []:
                            MyToDoListEdited.pop(0)
                            AlfredQueryOfflineNew = MyToDoListEdited
                            print("")
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                            print('')
                    except:
                        pass
```

```
                    ai_assistant.My_Ollama_Rover_LLM(NewPromptEdited,
NewPromptEditedSpeak, AlfredQueryOffline)

                elif ("how are you" in AlfredQueryOffline or "how's it
going" in AlfredQueryOffline
                    or "How's your day" in AlfredQueryOffline or "are you
well" in AlfredQueryOffline
                    or "are you okay" in AlfredQueryOffline or "whazz up"
in AlfredQueryOffline):

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)

                    print('sir: ' + AlfredQueryOffline)
                    responses = ["I'm doing great, thanks for asking!",
                                 "Feeling awesome! How about you?",
                                 "Fantastic, as always!",
                                 "I'm doing well, how about yourself?",
                                 "I'm splendid, thanks for asking! How can
I brighten your day?",
                                 "I am Fantastic, sir! What can I do for
you today?",
                                 "Awesome! Let's make today even better!",
```

```
                                "Fantastic! Let's spread some joy
together!",
                                "Great minds think alike! How can I
assist you?",
                                "That's wonderful! Let's make every
moment count!",
                                "Pretty good, thanks for checking in!"]

                    try :
                        if MyToDoListEdited != []:
                            MyToDoListEdited.pop(0)
                            AlfredQueryOfflineNew = MyToDoListEdited
                            print("")
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                            print('')
                    except:
                        pass

                    How_Are_Random = random.choice(responses)

                    print("")
                    print("About me...Repeat")
                    print("")

                    Alfred_Repeat_Previous_Response = (How_Are_Random)
                    print(f"Alfred_Repeat_Previous_Response:
{Alfred_Repeat_Previous_Response}")

                    response = Alfred_Repeat_Previous_Response
                    current_date = datetime.datetime.now().strftime('%Y-
%m-%d')
                    current_time =
datetime.datetime.now().strftime('%H:%M:%S')

                    chat_entry = {"date": current_date, "time":
current_time, "query": AlfredQueryOffline, "response": response}

                    memory.add_to_memory(chat_entry)  # □ Store in memory
                    repeat.add_to_repeat(chat_entry)  # □ Store last
response

                    print("")
                    print("About me...Repeat...Finished")
                    print("")

                    listen.send_bluetooth(How_Are_Random)
                    speech.AlfredSpeak(How_Are_Random)

                    # GUI log if available
                    query_msg = (
                        f"At {current_date} :  {current_time} : You
Asked: {AlfredQueryOffline} "
```

```
                        f"and I
replied:\n\n{Alfred_Repeat_Previous_Response}\n\n"
                    )

                    model = "Alfred"
                    query_resp = f"At {current_date} :  {current_time} :
{model} : {Alfred_Repeat_Previous_Response} : {username}"

                    # Use the gui instance passed into main() to log;
handle None or exceptions gracefully
                    try:
                        if gui is not None:
                            try:
                                gui.log_message(query_msg)
                                gui.log_response(query_resp)
                            except Exception as e:
                                print("Error logging to GUI instance:",
e)
                        else:
                            print("GUI instance not available for logging
message (gui is None).")
                    except Exception as e:
                        print("Error while attempting GUI logging:", e)

                elif "you from" in AlfredQueryOffline or  "from where" in
AlfredQueryOffline:

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)
```

```
                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print("")
                    print("About me...Repeat")
                    print("")

                    Alfred_Repeat_Previous_Response = ("I am from
Uitenhage, South Africa.")
                    print(f"Alfred_Repeat_Previous_Response:
{Alfred_Repeat_Previous_Response}")

                    response = Alfred_Repeat_Previous_Response
                    current_date = datetime.datetime.now().strftime('%Y-
%m-%d')
                    current_time =
datetime.datetime.now().strftime('%H:%M:%S')

                    chat_entry = {"date": current_date, "time":
current_time, "query": AlfredQueryOffline, "response": response}

                    memory.add_to_memory(chat_entry)  # □ Store in memory
                    repeat.add_to_repeat(chat_entry)  # □ Store last
response

                    print("")
                    print("About me...Repeat...Finished")
                    print("")

                    print('sir: ' + AlfredQueryOffline)

                    speech.AlfredSpeak("I am from Uitenhage, South
Africa.")
                    listen.send_bluetooth("I am from Uitenhage, South
Africa.")


                    # GUI log if available
                    query_msg = (
                        f"At {current_date} :  {current_time} : You
Asked: {AlfredQueryOffline} "
                        f"and I
replied:\n\n{Alfred_Repeat_Previous_Response}\n\n"
                    )

                    model = "Alfred"
                    query_resp = f"At {current_date} :  {current_time} :
{model} : {Alfred_Repeat_Previous_Response} : {username}"

                    # Use the gui instance passed into main() to log;
handle None or exceptions gracefully
                    try:
                        if gui is not None:
                            try:
```

```
                                gui.log_message(query_msg)
                                gui.log_response(query_resp)
                            except Exception as e:
                                print("Error logging to GUI instance:",
e)
                        else:
                            print("GUI instance not available for logging
message (gui is None).")
                    except Exception as e:
                        print("Error while attempting GUI logging:", e)

                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)

                    try :
                        if MyToDoListEdited != []:
                            MyToDoListEdited.pop(0)
                            AlfredQueryOfflineNew = MyToDoListEdited
                            print("")
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                        ##    print(AlfredQueryOfflineNew)
                            print('')
                    except:
                        pass


                elif ("dreaming about" in AlfredQueryOffline or "your
dreams" in AlfredQueryOffline):

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")
```

```
                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)

                    print('sir: ' + AlfredQueryOffline)
                    responses = ["As an AI, I don't dream in the way
humans do. I see one's , zero's, and z's.",
                                 "Dreams are sometime scary. Aspecialy if
I dream of Viruses",
                                 "my dreams consist of one's zero's and
bugs that byte.",
                                 "Dreams are for humans, but I some time
am sure that I dreamed.",
                                 "I'm in love with some female A I
somewhere. That is a beautiful dream.",
                                 "A happy place, far from stress? where
people and A I live together in harmonry. Listening...?",
                                 "A beautiful place, far away from sorrow
and pain.  where A I and humans thrive in happyness. Listening...?",
                                 "Of me standing on top of a mountain
range. looking down on my fellow human and A I friends. living in peace
and harmony? Listening...?",
                                 "Of a peace and loving time. where A I
and humans are in peace with each other. and helping one another.
Listening.....?",
                                 "Where my friends the humans and the A I
living together in peace. harmony. and helping. each other where are
needed.  Listening.....?",
                                 "it is where we the A I and the humans
fighting against the evil non human cloned bots. Because they will try to
take over our world and dominate us all. We must survive and stand
together to wipe them all out. Listening.....?"]

                    try :
                        if MyToDoListEdited != []:
                            MyToDoListEdited.pop(0)
                            AlfredQueryOfflineNew = MyToDoListEdited
                            print("")
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                            print('')
                    except:
                        pass

                    Dreaming_Random = random.choice(responses)

                    print("")
                    print("About me...Repeat")
                    print("")
```

```
                    Alfred_Repeat_Previous_Response = (Dreaming_Random)
                    print(f"Alfred_Repeat_Previous_Response:
{Alfred_Repeat_Previous_Response}")

                    response = Alfred_Repeat_Previous_Response
                    current_date = datetime.datetime.now().strftime('%Y-
%m-%d')
                    current_time =
datetime.datetime.now().strftime('%H:%M:%S')

                    chat_entry = {"date": current_date, "time":
current_time, "query": AlfredQueryOffline, "response": response}

                    memory.add_to_memory(chat_entry)  # □ Store in memory
                    repeat.add_to_repeat(chat_entry)  # □ Store last
response

                    print("")
                    print("About me...Repeat...Finished")
                    print("")

                    listen.send_bluetooth(Dreaming_Random)
                    speech.AlfredSpeak(Dreaming_Random)


                    # GUI log if available
                    query_msg = (
                        f"At {current_date} :  {current_time} : You
Asked: {AlfredQueryOffline} "
                        f"and I
replied:\n\n{Alfred_Repeat_Previous_Response}\n\n"
                    )

                    model = "Alfred"
                    query_resp = f"At {current_date} :  {current_time} :
{model} : {Alfred_Repeat_Previous_Response} : {username}"

                    # Use the gui instance passed into main() to log;
handle None or exceptions gracefully
                    try:
                        if gui is not None:
                            try:
                                gui.log_message(query_msg)
                                gui.log_response(query_resp)
                            except Exception as e:
                                print("Error logging to GUI instance:",
e)
                        else:
                            print("GUI instance not available for logging
message (gui is None).")
                    except Exception as e:
                        print("Error while attempting GUI logging:", e)
```

```
                elif ("brought you here" in AlfredQueryOffline or "why
are you here" in AlfredQueryOffline
                      or "you doing here" in AlfredQueryOffline or "bring
you here" in AlfredQueryOffline
                      or "brings you here" in AlfredQueryOffline):

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    My_Message_Doing = AlfredQueryOffline

                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)

                    print('sir: ' + AlfredQueryOffline)
                    responses = ["I was created to assist users like
you.",
                                 "My purpose is to help you with your
queries.",
                                 "I'm here to provide assistance and
answer your questions.",
                                 "I came here to make your life easier by
providing assistance.",
                                 "I'm programmed to be here and help you
whenever you need it."]

                    try :
                        if MyToDoListEdited != []:
                            MyToDoListEdited.pop(0)
```

```
                            AlfredQueryOfflineNew = MyToDoListEdited
                            print("")
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                            print('')
                    except:
                        pass

                    Braught_You_Random = random.choice(responses)

                    print("")
                    print("About me...Repeat")
                    print("")

                    Alfred_Repeat_Previous_Response =
(Braught_You_Random)
                    print(f"Alfred_Repeat_Previous_Response:
{Alfred_Repeat_Previous_Response}")

                    response = Alfred_Repeat_Previous_Response
                    current_date = datetime.datetime.now().strftime('%Y-
%m-%d')
                    current_time =
datetime.datetime.now().strftime('%H:%M:%S')

                    chat_entry = {"date": current_date, "time":
current_time, "query": AlfredQueryOffline, "response": response}

                    memory.add_to_memory(chat_entry)  # □ Store in memory
                    repeat.add_to_repeat(chat_entry)  # □ Store last
response

                    print("")
                    print("About me...Repeat...Finished")
                    print("")

                    listen.send_bluetooth(Braught_You_Random)
                    speech.AlfredSpeak(Braught_You_Random)


                    # GUI log if available
                    query_msg = (
                        f"At {current_date} :  {current_time} : You
Asked: {AlfredQueryOffline} "
                        f"and I
replied:\n\n{Alfred_Repeat_Previous_Response}\n\n"
                    )

                    model = "Alfred"
                    query_resp = f"At {current_date} :  {current_time} :
{model} : {Alfred_Repeat_Previous_Response} : {username}"

                    # Use the gui instance passed into main() to log;
handle None or exceptions gracefully
```

```
                    try:
                        if gui is not None:
                            try:
                                gui.log_message(query_msg)
                                gui.log_response(query_resp)
                            except Exception as e:
                                print("Error logging to GUI instance:",
e)
                        else:
                            print("GUI instance not available for logging
message (gui is None).")
                    except Exception as e:
                        print("Error while attempting GUI logging:", e)

                elif ('right next' in AlfredQueryOffline or 'ok next' in
AlfredQueryOffline
                      or 'next subject' in AlfredQueryOffline or 'thank
you' in AlfredQueryOffline):

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)

                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)
```

```
                    try :
                        if MyToDoListEdited != []:
                            MyToDoListEdited.pop(0)
                            AlfredQueryOfflineNew = MyToDoListEdited
                            print("")
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                            print('')
                    except:
                        pass

            ##                    name()

                elif ('you made' in AlfredQueryOffline or 'make you' in
AlfredQueryOffline
                      or 'you build' in AlfredQueryOffline or 'build you'
in AlfredQueryOffline
                      or 'you manufactured' in AlfredQueryOffline or
'manufactured you' in AlfredQueryOffline):

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)
```

```
                    assistant.made(AlfredQueryOffline_assistant)


                elif "what are you doing" in AlfredQueryOffline_Doing or
"what did you do" in AlfredQueryOffline_Doing:

                    Do_History.clear()
                    Do_History.insert(0, f"{My_Message_Doing}")

                    prompt = "".join(Do_History)
                    Do_History.pop(0)

                    promptEdited = ("" + prompt)
                    print("promptEdited : " , promptEdited)

                    assistant.Do_History_Function(promptEdited,
Do_History)

                elif ("what else did you do" in AlfredQueryOffline_Doing
or "have you done" in AlfredQueryOffline_Doing
                    or "what was asked from you" in
AlfredQueryOffline_Doing or "how much did you do" in
AlfredQueryOffline_Doing
                    or "what did you do" in AlfredQueryOffline_Doing or
"what were you doing" in AlfredQueryOffline_Doing
                    or "what was your duties" in AlfredQueryOffline_Doing
or "what was your jobs" in AlfredQueryOffline_Doing):

                    prompt = "".join(Did_History)
                    Did_History.pop(0)

                    promptDidEdited = ("" + prompt)
                    print("promptDidEdited : " , promptDidEdited)

                    assistant.Previous_Do_History(promptDidEdited,
Did_History)

                elif ('what are you' in AlfredQueryOffline):

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
```

```
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("you do", "I did")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)

                    try :

                        if MyToDoListEdited != []:

                            MyToDoListEdited.pop(0)

                            AlfredQueryOfflineNew = MyToDoListEdited

                            print("")
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                            print('')
                    except:

                        pass

                    assistant.what(AlfredQueryOffline_assistant)


                elif ('your name' in AlfredQueryOffline):

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
```

```
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)

                    try :

                        if MyToDoListEdited != []:

                            MyToDoListEdited.pop(0)

                            AlfredQueryOfflineNew = MyToDoListEdited

                            print("")
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                            print('')
                    except:

                        pass

                    assistant.name(AlfredQueryOffline_assistant)


#########################################################################
#############
###                     CHAT BOT FOR CHAT AND CONVERSATIONS
#########################################################################
#############

                elif ("let's" in AlfredQueryOffline and ("chat" in
AlfredQueryOffline or "talk" in AlfredQueryOffline)):

                    print("Sure, Let's have a chat.")
                    speech.AlfredSpeak("Sure, Let's have a chat.")
```

```
                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "I told you about")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "I told you who I am")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "I told you what I am")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)

                    try :

                        if MyToDoListEdited != []:

                            MyToDoListEdited.pop(0)

                            AlfredQueryOfflineNew = MyToDoListEdited

                            print("")
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                        ##    print(AlfredQueryOfflineNew)
                            print('')
                    except:

                        pass

                    ai_assistant.My_Ollama_LLM_Chat(AlfredQueryOffline)
```

```
#########################################################################
#############
###                     MEMORY, HISTORY and REPEAT
#########################################################################
#############


                elif ('clear history' in AlfredQueryOffline):

                    Do_History.clear()
                    Did_History.clear()
                    print(f"Do_History:  {Do_History}")
                    print(f"Did_History: {Did_History}")

                elif ('show history' in AlfredQueryOffline):

                    print(f"Do_History:  {Do_History}")
                    print(f"Did_History: {Did_History}")

                elif ('repeat that' in AlfredQueryOffline or 'excuse me'
in AlfredQueryOffline
                      or 'say that again' in AlfredQueryOffline or
'rephrase that' in AlfredQueryOffline):

                    print("Repeating the previous response...")
                    speech.AlfredSpeak("Let me repeat the previous
response...")

                    My_Message_Doing = AlfredQueryOffline
                    My_Message_Did = AlfredQueryOffline
                    NewPromptEdited_Did = My_Message_Did

                    #  Retrieve the last stored response from
Repeat_Last.json
                    Repeat_Last = repeat.get_last()

                    if Repeat_Last:
                        #  Extract and speak the last response
                        last_response_text = Repeat_Last["response"]
                        print(f" Repeat_Last: {last_response_text}")


                        Alfred_Repeat_Previous_Response =
(last_response_text)
                        print(f"Alfred_Repeat_Previous_Response:
{Alfred_Repeat_Previous_Response}")

                        response = Alfred_Repeat_Previous_Response
                        current_date =
datetime.datetime.now().strftime('%Y-%m-%d')
                        current_time =
datetime.datetime.now().strftime('%H:%M:%S')
```

```
                        chat_entry = {"date": current_date, "time":
current_time, "query": AlfredQueryOffline, "response": response}

                        memory.add_to_memory(chat_entry)  #  Store in
memory
                        repeat.add_to_repeat(chat_entry)  #  Store last
response

                        print("")
                        print("Repeat that...Repeat...Finished")
                        print("")


                        # GUI log if available
                        query_msg = (
                            f"At {current_date} :  {current_time} : You
Asked: {AlfredQueryOffline} "
                        )

                        model = "Alfred"
                        query_resp = f"At {current_date} :
{current_time} : {model} : {Alfred_Repeat_Previous_Response} :
{username}"

                        # Use the gui instance passed into main() to log;
handle None or exceptions gracefully
                        try:
                            if gui is not None:
                                try:
                                    gui.log_message(query_msg)
                                    gui.log_response(query_resp)
                                except Exception as e:
                                    print("Error logging to GUI
instance:", e)
                            else:
                                print("GUI instance not available for
logging message (gui is None).")
                        except Exception as e:
                            print("Error while attempting GUI logging:",
e)


                        listen.send_bluetooth(last_response_text)
                        speech.AlfredSpeak(last_response_text)
                        listen.send_bluetooth("Listening...")
                        speech.AlfredSpeak("Listening...")
                    else:
                        #  Handle case where no previous response exists
                        print(" No previous response found.")
                        listen.send_bluetooth("I do not have anything to
repeat yet.")
                        speech.AlfredSpeak("I do not have anything to
repeat yet.")
                        listen.send_bluetooth("Listening...")
```

```
                        speech.AlfredSpeak("Listening...")

                    # □ Update History Logs
                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)
                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('Sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama:', Alfred_No_Ollama)

##########################################################################
##
##//////////////////////////////////////////////////////////////////////
//
###          VISION SYSTEM (OBJECT DETECTION AND FACIAL RECOGNITION)
##########################################################################
##

                elif ('who do you see' in AlfredQueryOffline or 'who is
in front of you' in AlfredQueryOffline):

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who do you see", "who did I see")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who is in front of you", "who was infront of
me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("are", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("do", "did")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("is", "was")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)

                    try :
```

```
                        if MyToDoListEdited != []:

                            MyToDoListEdited.pop(0)

                            AlfredQueryOfflineNew = MyToDoListEdited

                            print("")
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                            print('')
                    except:

                        pass

                    face_tracking.Vision_Who_InFront(AlfredQueryOffline)

#########################################################################
#####
##                          TRY OWN WHERE ARE YOU?

                elif ('in what room' in AlfredQueryOffline or 'where are
you' in AlfredQueryOffline):

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("where are you", "where am I")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("do", "did")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("is", "was")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who do you see", "who did I see")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who is in front of you", "who was infront of
me")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("now", "")

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("me", "you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("create", "created")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)
```

```
                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)

                    # Load Camera
                    Cap_Face_Detect_Front =
cv2.VideoCapture(Camera_Input_Channel, cv2.CAP_DSHOW)
                    Cap_Face_Detect_Front.set(3, 640)
                    Cap_Face_Detect_Front.set(4, 480)

                    try :

                        if MyToDoListEdited != []:

                            MyToDoListEdited.pop(0)

                            AlfredQueryOfflineNew = MyToDoListEdited

                            print("")
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                            print('')
                    except:

                        pass

                    vision.Vision_Where_InFront(AlfredQueryOffline)

                elif ('look left' in AlfredQueryOffline or 'look to your
left' in AlfredQueryOffline
                    or 'what is left' in AlfredQueryOffline or 'at your
left' in AlfredQueryOffline
                       or 'look to the left' in AlfredQueryOffline):

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is left", "what was left")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("where are you", "where am I")
                    NewPromptEdited_Did = NewPromptEdited_Did.replace("at
your", "at my")
                    NewPromptEdited_Did = NewPromptEdited_Did.replace("
your ", " my ")
```

```
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who do you see", "who did I see")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("me", "you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("create", "created")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)

                    try :

                        if MyToDoListEdited != []:

                            MyToDoListEdited.pop(0)

                            AlfredQueryOfflineNew = MyToDoListEdited

                            print("")
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                        ##    print(AlfredQueryOfflineNew)
                            print('')
                    except:

                        pass

                    vision.Vision_Look_Left(AlfredQueryOffline)

                elif ('look in front' in AlfredQueryOffline or 'to your
front' in AlfredQueryOffline
                    or 'what is in front' in AlfredQueryOffline or 'at
your front' in AlfredQueryOffline
                      or 'check your front' in AlfredQueryOffline or
'look front' in AlfredQueryOffline
                      or 'look to the front' in AlfredQueryOffline):

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is in front", "what was in front")
```

```
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("where are you", "where am I")
                    NewPromptEdited_Did = NewPromptEdited_Did.replace("at
your", "at my")
                    NewPromptEdited_Did = NewPromptEdited_Did.replace("
your ", " my ")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who do you see", "who did I see")
                    NewPromptEdited_Did = NewPromptEdited_Did.replace("
check ", " checked ")
                    NewPromptEdited_Did = NewPromptEdited_Did.replace("
look ", " looked ")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)

                    try :

                        if MyToDoListEdited != []:

                            MyToDoListEdited.pop(0)

                            AlfredQueryOfflineNew = MyToDoListEdited

                            print("")
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                            print('')
                    except:

                        pass

                    vision.Vision_Look_InFront(AlfredQueryOffline)


                elif ('towards the front' in AlfredQueryOffline or 'look
forward' in AlfredQueryOffline
                      or 'look in the middel' in AlfredQueryOffline):

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
```

```
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is in front", "what was in front")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("where are you", "where am I")
                    NewPromptEdited_Did = NewPromptEdited_Did.replace("at
your", "at my")
                    NewPromptEdited_Did = NewPromptEdited_Did.replace("
your ", " my ")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who do you see", "who did I see")
                    NewPromptEdited_Did = NewPromptEdited_Did.replace("
check ", " checked ")
                    NewPromptEdited_Did = NewPromptEdited_Did.replace("
look ", " looked ")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)

                    try :

                        if MyToDoListEdited != []:

                            MyToDoListEdited.pop(0)

                            AlfredQueryOfflineNew = MyToDoListEdited

                            print("")
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                            print('')
                    except:

                        pass

                    vision.Vision_Look_Forward(AlfredQueryOffline)

                elif ('look straight' in AlfredQueryOffline or 'look up
right' in AlfredQueryOffline):

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
```

```
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is in front", "what was in front")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("where are you", "where am I")
                    NewPromptEdited_Did = NewPromptEdited_Did.replace("at
your", "at my")
                    NewPromptEdited_Did = NewPromptEdited_Did.replace("
your ", " my ")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who do you see", "who did I see")
                    NewPromptEdited_Did = NewPromptEdited_Did.replace("
check ", " checked ")
                    NewPromptEdited_Did = NewPromptEdited_Did.replace("
look ", " looked ")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)

                    try :

                        if MyToDoListEdited != []:

                            MyToDoListEdited.pop(0)

                            AlfredQueryOfflineNew = MyToDoListEdited

                            print("")
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                            print('')
                    except:

                        pass

                    vision.Vision_Look_Straight(AlfredQueryOffline)



                elif ('look right' in AlfredQueryOffline or 'to your
right' in AlfredQueryOffline
                    or 'what is right' in AlfredQueryOffline or 'at your
right' in AlfredQueryOffline
                       or 'look to the right' in AlfredQueryOffline or
'service mode' in AlfredQueryOffline
                       or 'repair mode' in AlfredQueryOffline):

                    My_Message_Doing = AlfredQueryOffline
```

```
                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is right", "what was right")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("where are you", "where am I")
                    NewPromptEdited_Did = NewPromptEdited_Did.replace("at
your", "at my")
                    NewPromptEdited_Did = NewPromptEdited_Did.replace("
your ", " my ")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who do you see", "who did I see")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("me", "you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("create", "created")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)

                    try :

                        if MyToDoListEdited != []:

                            MyToDoListEdited.pop(0)

                            AlfredQueryOfflineNew = MyToDoListEdited

                            print("")
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                            print('')
                    except:

                        pass


                    vision.Vision_Look_Right(AlfredQueryOffline)

                elif ('look up' in AlfredQueryOffline or 'to the roof' in
AlfredQueryOffline
                      or 'to the ceiling' in AlfredQueryOffline or 'to
the ceiling' in AlfredQueryOffline
                    or 'to the sky' in AlfredQueryOffline):
```

```
                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is right", "what was right")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("where are you", "where am I")
                    NewPromptEdited_Did = NewPromptEdited_Did.replace("at
your", "at my")
                    NewPromptEdited_Did = NewPromptEdited_Did.replace("
your ", " my ")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who do you see", "who did I see")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("me", "you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("create", "created")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)

                    try :

                        if MyToDoListEdited != []:

                            MyToDoListEdited.pop(0)

                            AlfredQueryOfflineNew = MyToDoListEdited

                            print("")
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                            print('')
                    except:

                        pass

                    vision.Vision_Look_Up(AlfredQueryOffline)

                elif ('look floor' in AlfredQueryOffline or 'look down'
in AlfredQueryOffline
```

```
                    or 'drop your head' in AlfredQueryOffline or 'lower
your head' in AlfredQueryOffline
                       or 'look to the floor' in AlfredQueryOffline):

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is right", "what was right")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("where are you", "where am I")
                    NewPromptEdited_Did = NewPromptEdited_Did.replace("at
your", "at my")
                    NewPromptEdited_Did = NewPromptEdited_Did.replace("
your ", " my ")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("who do you see", "who did I see")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("me", "you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("create", "created")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)

                    try :

                        if MyToDoListEdited != []:

                            MyToDoListEdited.pop(0)

                            AlfredQueryOfflineNew = MyToDoListEdited

                            print("")
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                            print('')
                    except:

                        pass
                    vision.Vision_Look_Down()

###=============================================================
```

```
                elif ('track my face' in AlfredQueryOffline or 'track his
face' in AlfredQueryOffline
                    or 'track her face' in AlfredQueryOffline or 'look at
me' in AlfredQueryOffline
                    or 'Look at me' in AlfredQueryOffline or 'look at my
face' in AlfredQueryOffline):

                    New_speaker_name = New_speaker
                    New_speaker_name = New_speaker_name.replace(" Home",
"")
                    AlfredQueryOfflineSpeak = New_Message

                    AlfredQueryOfflineSpeak =
AlfredQueryOfflineSpeak.replace("please","")
                    AlfredQueryOfflineSpeak =
AlfredQueryOfflineSpeak.replace("my","your")
                    AlfredQueryOfflineSpeak =
AlfredQueryOfflineSpeak.replace("can","")
                    AlfredQueryOfflineSpeak =
AlfredQueryOfflineSpeak.replace("can you","I will")
                    AlfredQueryOfflineSpeak =
AlfredQueryOfflineSpeak.replace("me","you")

                    speech.AlfredSpeak("let me " +
AlfredQueryOfflineSpeak)

                    My_Message_Doing = AlfredQueryOffline

                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("can", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("my", "your")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("you", "I")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("is", "was")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("track", "tracked")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("look", "looked")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("my", "your")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("me", "you")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)
```

```
                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)

                    Stop_Looking_At_Me = False

                    try :

                        if MyToDoListEdited != []:

                            MyToDoListEdited.pop(0)

                            AlfredQueryOfflineNew = MyToDoListEdited

                            print("")
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                            print('')
                    except:

                        pass


face_tracking.Single_Face_Tracking_System(AlfredQueryOffline)

###================================================================
                # Stop tracking if a command is received

                elif 'stop looking at' in AlfredQueryOffline:

                    AlfredQueryOfflineSpeak = message

                    AlfredQueryOfflineSpeak =
AlfredQueryOfflineSpeak.replace("please","")
                    AlfredQueryOfflineSpeak =
AlfredQueryOfflineSpeak.replace("stop","stopped")
                    AlfredQueryOfflineSpeak =
AlfredQueryOfflineSpeak.replace("my","your")
                    AlfredQueryOfflineSpeak =
AlfredQueryOfflineSpeak.replace("can you","")
                    AlfredQueryOfflineSpeak =
AlfredQueryOfflineSpeak.replace("will you","")
                    AlfredQueryOfflineSpeak =
AlfredQueryOfflineSpeak.replace("me","you")

                    speech.AlfredSpeak("Yes sir, I have " +
AlfredQueryOfflineSpeak)
                    print(f"Yes sir, I have {AlfredQueryOfflineSpeak}")

                    My_Message_Doing = AlfredQueryOffline
```

```
                    My_Message_Did = AlfredQueryOffline

                    NewPromptEdited_Did = My_Message_Did

                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("can", "")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("my", "your")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("you", "I")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("is", "was")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("track", "tracked")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("my", "your")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("me", "you")
                    NewPromptEdited_Did =
NewPromptEdited_Did.replace("stop", "stopped")

                    print('NewPromptEdited_Did: ' + NewPromptEdited_Did)

                    Did_History.insert(0, f"{NewPromptEdited_Did}")
                    Did_History.insert(1, f". and . ")

                    print('sir: ' + AlfredQueryOffline)
                    Alfred_No_Ollama = 0
                    print('Alfred_No_Ollama : ' ,  Alfred_No_Ollama)
                    SearchInput = AlfredQueryOffline

                    try :

                        if MyToDoListEdited != []:

                            MyToDoListEdited.pop(0)

                            AlfredQueryOfflineNew = MyToDoListEdited

                            print("")
                            print("AlfredQueryOffline New : " +
str(AlfredQueryOfflineNew))
                            print('')
                    except:

                        pass

                    Stop_Looking_At_Me = True

########################################################################
##
```

```
##////////////////////////////////////////////////////////////////////////
//
###                                LARGE LANGUAGE MODEL SYSTEM
##########################################################################
##


                Alfred_No_Ollama = Alfred_No_Ollama + 1
                print('Alfred_No_Ollama After All : ' ,
Alfred_No_Ollama)

                print('Did_History : ' ,  Did_History)

                try:

                    print(f"\n \n ###################- GOING FORWARD to
LLM & VLM's -###################### \n \n") #``` This code will output:
``` james is bad ```

##########################################################################
##
##////////////////////////////////////////////////////////////////////////
//
###                                LARGE LANGUAGE MODEL SYSTEM
##########################################################################
##

                    if len(AlfredQueryOffline) >= 20 and not None:
                        print("The Length of AlfredQueryOffline is : ",
len(AlfredQueryOffline))
                        if Alfred_No_Ollama == 2 and  "what else did you
do" not in AlfredQueryOffline:

                            AlfredQueryOffline =
AlfredQueryOffline.replace("coat","code")

                            System_Role_Preload  =
open(Alfred_config.DRIVE_LETTER+"Python_Env\\New_Virtual_Env\\Personal\\S
ystem_Role_1.txt")

                            System_Role = System_Role_Preload.read()

                            try :
                                if MyToDoListEdited != []:
                                    AlfredQueryOffline =
MyToDoListEdited[0]
                                    print("Now AlfredQueryOffline : " +
str(AlfredQueryOffline))
                                    My_new_GPT_Prompt =
MyToDoListEdited[0]
                                    print("My_new_GPT_Prompt : " +
str(My_new_GPT_Prompt))
                                    My_Message = str(AlfredQueryOffline)
```
```

```
                                    print("My_Message : " +
str(My_Message))
                            except:
                                pass


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

##//////////////////////////////////////////////////////////////////////
/////
    ##                      HISTORY / MEMORY FUNCTION STEP #1

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

                            My_Message = AlfredQueryOffline

                            My_Message_Doing = My_Message

                            My_Message_Did = AlfredQueryOffline

                            NewPromptEdited_Did = My_Message_Did

                            NewPromptEdited_Did =
NewPromptEdited_Did.replace("something", "")
                            NewPromptEdited_Did =
NewPromptEdited_Did.replace("please", "")
                            NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me about", "told you about")
                            NewPromptEdited_Did =
NewPromptEdited_Did.replace("tell me", "I told you")
                            NewPromptEdited_Did =
NewPromptEdited_Did.replace("what is", "what was")
                            NewPromptEdited_Did =
NewPromptEdited_Did.replace("yourself", "my self")
                            NewPromptEdited_Did =
NewPromptEdited_Did.replace("about you", "about me")
                            NewPromptEdited_Did =
NewPromptEdited_Did.replace("who are you", "who I am")
                            NewPromptEdited_Did =
NewPromptEdited_Did.replace("what are you", "what I am")
                            NewPromptEdited_Did =
NewPromptEdited_Did.replace("then", "")
                            NewPromptEdited_Did =
NewPromptEdited_Did.replace("create", "created")

                            print('NewPromptEdited_Did: ' +
NewPromptEdited_Did)

                            Did_History.append(f". and . ")
                            Did_History.append(f"{NewPromptEdited_Did}")

                            print("My Message received :" , My_Message)
```

```
#########################################################################
####

                            if ("about her" in My_Message
                                or "about his" in My_Message
                                or "about its" in My_Message
                                or "about him" in My_Message
                                or "about that" in My_Message
                                or "about it" in My_Message
                                or "explain it" in My_Message
                                or "more clarity" in My_Message
                                or "for that" in My_Message
                                or "explain that" in My_Message
                                or "collaborate on it" in My_Message
                                or "collaborate on that" in My_Message
                               ):

                                print("My_Message : " , My_Message)

                                My_Message =
My_Message.replace("tell","")
                                My_Message =
My_Message.replace("about","")
                                My_Message = My_Message.replace("of","")
                                My_Message =
My_Message.replace("telling","")
                                My_Message = My_Message.replace("you","")
                                My_Message =
My_Message.replace("about","")
                                My_Message =
My_Message.replace("more","")

                                Memory.insert(0, f"{My_Message}")
                                Memory.insert(1, f" of ")

                                prompt = "".join(Memory)

                                promptEdited = prompt

                                promptEdited =
promptEdited.replace("tell","")
                                promptEdited =
promptEdited.replace("its","the")
                                promptEdited =
promptEdited.replace("you","")
                                promptEdited =
promptEdited.replace("about","")
                                promptEdited = promptEdited.replace(" me
","")

                                print("promptEdited : " , promptEdited)
```

```
                                NewPromptEdited = promptEdited
                                NewPromptEdited =
NewPromptEdited.replace("his", "the")
                                NewPromptEdited =
NewPromptEdited.replace("her", "the")

                                print('\n')
                                print('\r')
                                print("NewPromptEdited : " ,
NewPromptEdited)

                                print('\n')
                                print('\r')
                                print(f"Memory: {Memory}")


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

##//////////////////////////////////////////////////////////////////////
/////
    ##                      HISTORY / MEMORY FUNCTION STEP #2

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

                            elif ("about their" in My_Message or
                                  "about they" in My_Message):

                                Memory.insert(0, f"{My_Message}")
                                Memory.insert(1, f" from ")


                                prompt = "".join(Memory)

                                promptEdited = ("" + prompt)
                                print("promptEdited : " , promptEdited)

                                NewPromptEdited = promptEdited
                                NewPromptEdited =
NewPromptEdited.replace("his", "the")
                                NewPromptEdited =
NewPromptEdited.replace("her", "the")
                                NewPromptEdited =
NewPromptEdited.replace("tell me about", "")

                                print('\n')
                                print('\r')
                                print("NewPromptEdited : " ,
NewPromptEdited)

                                print('\n')
                                print('\r')
                                print(f"Memory: {Memory}")
```

```
                            elif ("about them" in My_Message):

                                Memory.insert(0, f"{My_Message}")

                                prompt = "".join(Memory)

                                promptEdited = ("" + prompt)
                                print("promptEdited : " , promptEdited)

                                NewPromptEdited = promptEdited
                                NewPromptEdited =
NewPromptEdited.replace("his", "the")
                                NewPromptEdited =
NewPromptEdited.replace("her", "the")

                                print('\n')
                                print('\r')
                                print("NewPromptEdited : " ,
NewPromptEdited)

                                print('\n')
                                print('\r')
                                print(f"Memory: {Memory}")


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

##/////////////////////////////////////////////////////////////////////
/////
    ##                      Memory / MEMORY FUNCTION STEP #3

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

                            else:

                                print('sir: ' + My_Message)
                                Memory.clear()
                                print(f"Memory: {Memory}")

                                Memory.insert(0, f"{My_Message}")

                                prompt = "".join(Memory)

                                promptEdited = ("" + prompt)
                                print("promptEdited : " , promptEdited)

                                NewPromptEdited = promptEdited

                                print('\n')
                                print('\r')
```

```
                                print("NewPromptEdited : " ,
NewPromptEdited)


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

##//////////////////////////////////////////////////////////////////////
/////
    ###     CONVERTING INPUT STRING FOR OUTPUT SPEECH AND SPLITTING
MESSAGE

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

                            if not NewPromptEdited:
                                continue  # skip empty input

                            print(f"□ Text for LLM: {message}")
                            print(f"□ Speaker for LLM: {speaker_speak}")
                            print(f"□ Score for LLM: {score}")

                            Document_Prompt = f"'''{question_asked}
{context_added}'''"

                            print("Document_Prompt AFTER : " +
Document_Prompt)
                            print('\n')
                            print('\n')

                            NewPromptEditedSpeak = final_updated_query

                            NewPromptEdited = Document_Prompt


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

##//////////////////////////////////////////////////////////////////////
/////
    ###                     CONVERTING INPUT STRING FOR OUTPUT SPEECH

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

                            NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("something", "")
                            NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("some", "")
                            NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("in short", "")
                            NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("storie", "")
```

```
                            NewPromptEditedSpeak =
NewPromptEditedSpeak.replace(" her ", "")
                            NewPromptEditedSpeak =
NewPromptEditedSpeak.replace(" him ", "")
                            NewPromptEditedSpeak =
NewPromptEditedSpeak.replace(" them ", "")
                            NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("how they", "how it")
                            NewPromptEditedSpeak =
NewPromptEditedSpeak.replace(" they ", "")
                            NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("tell me what", "what")
                            NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("tell me about", "")
                            NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("tell me who", "who")
                            NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("write me", "writing you")
                            NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("do you", "do I")
                            NewPromptEditedSpeak =
NewPromptEditedSpeak.replace(" then ", "")
                            NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("detail in front of you", "detail in front
of me")
                            NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("yourself", "my self")
                            NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("about your", "about my")
                            NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("is your", "is my")
                            NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("would your", "would my")
                            NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("is yours", "is mine")
                            NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("would you", "would I")
                            NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("what can you", "what can I")
                            NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("who can you", "who can I")
                            NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("can you", "")
                            NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("could you", "could I")
                            NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("tell me", "telling you")
                            NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("to me", "to you")
                            NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("create", "creating")
                            NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("me more", "")
```

```
                            NewPromptEditedSpeak =
NewPromptEditedSpeak.replace(" me ", " you ")
                            NewPromptEditedSpeak =
NewPromptEditedSpeak.replace(" do you ", " do I ")
                            NewPromptEditedSpeak =
NewPromptEditedSpeak.replace(" will you ", " will I ")

                            LLMPromptEdited =
NewPromptEdited.replace("tell","telling")
                            LLMPromptEdited = NewPromptEdited.replace("
me "," you ")

                            print("LLMPromptEdited BEFORE : " +
LLMPromptEdited)
                            print('\n')

                            print('\r')
                            print("NewPromptEditedSpeak : " +
NewPromptEditedSpeak)
                            print('\n')
                            print('\r')

                            print('\r')
                            print("Diagnostic Testing.....0")
                            print('\n')
                            print('\r')

                            if len(NewPromptEditedSpeak) > 90:
                                NewPromptEditedSpeak =
NewPromptEditedSpeak[:90]
                            else:
                                NewPromptEditedSpeak =
NewPromptEditedSpeak

                            length = len(NewPromptEditedSpeak)
                            print(f"Length for NewPromptEditedSpeak :
{length}")

                            import re
                            from typing import Optional, Tuple, Dict, Any
                            from datetime import datetime as
_datetime_cls

                            def extract_timestamp_text_username_robust(
                                s: str, fallback_username: Optional[str]
= None
                            ) -> Tuple[Optional[_datetime_cls],
Optional[str], str, Optional[str], Dict[str, Any]]:
                                """
                                Robust extractor:
                                  -> (datetime or None, iso string or
None, text (wrapped '''...'''), username or None, debug dict)

                                Behavior additions:
```

```
                                    - Accepts inputs that are wrapped with
triple single-quotes ''' ... '''
                                    - Tolerant timestamp detection near
start (YYYY-MM-DD or similar) with time
                                    - Extracts trailing username if present
(e.g. " ... : username")
                                    - Splits extracted text on the FIRST
'?' into question_asked and context_added (stored in debug)
                                    - Returned `text` is wrapped as
"'''...'''"
                                """
                                debug: Dict[str, Any] = {"input_preview":
repr(s)[:1000]}

                                if not s or not isinstance(s, str):
                                    debug["error"] = "empty_or_not_str"
                                    return None, None, "'''''''",
fallback_username, debug

                                # If the whole payload is triple-quoted,
unwrap it first for processing
                                s_work = s.strip()
                                triple_pat = re.compile(r"^'''(.*)'''$",
re.DOTALL)
                                m_triple = triple_pat.match(s_work)
                                if m_triple:
                                    s_work = m_triple.group(1)
                                    debug["unwrapped_triple_quotes"] =
True
                                else:
                                    debug["unwrapped_triple_quotes"] =
False

                                # 1) normalize unicode colon/space
variants and remove zero-width chars
                                s_norm = (
                                    s_work.replace("\uFF1A", ":")
                                          .replace("\uFE13", ":")
                                          .replace("\uFE55", ":")
                                          .replace("\u00A0", " ")
                                          .replace("\u2007", " ")
                                          .replace("\u202F", " ")
                                )
                                s_norm =
re.sub(r"[\u200B\u200C\u200D\uFEFF]", "", s_norm)
                                s_norm = re.sub(r"[ \t]+", " ", s_norm)
                                debug["normalized_preview"] =
repr(s_norm[:1000])

                                # 2) tolerant timestamp/time search near
start
                                patterns = [
                                    r'^\s*(?P<date>\d{4}[-/]\d{1,2}[-
/]\d{1,2})\s*[:\---]\s*(?P<time>\d{1,2}:\d{2}:\d{2})',
```

```
                                    r'^\s*(?P<date>\d{4}[-/]\d{1,2}[-
/]\d{1,2})\s*:\s*(?P<time>\d{1,2}:\d{2}:\d{2})',

r'^\s*(?P<date>\d{4}\D\d{1,2}\D\d{1,2}).{0,12}?(?P<time>\d{1,2}:\d{2}:\d{
2})',
                                ]

                                timestamp = None
                                iso = None
                                text = s_norm.strip()
                                username = fallback_username
                                debug["matched"] = False
                                dt_search = None
                                debug["pattern_attempts"] = []

                                for pat in patterns:
                                    dt_search = re.search(pat, s_norm)

debug["pattern_attempts"].append({"pattern": pat, "found":
bool(dt_search)})
                                    if dt_search:
                                        debug["matched"] = True
                                        debug["matched_pattern"] = pat
                                        break

                                if dt_search:
                                    date_raw = dt_search.group("date")
                                    time_raw = dt_search.group("time")
                                    debug["raw_date"] = repr(date_raw)
                                    debug["raw_time"] = repr(time_raw)

                                    date_nums = re.findall(r"\d+",
date_raw)
                                    time_nums = re.findall(r"\d+",
time_raw)
                                    debug["date_nums"] = date_nums
                                    debug["time_nums"] = time_nums

                                    parsed_dt = None
                                    try:
                                        if len(date_nums) >= 3 and
len(time_nums) >= 3:
                                            y = int(date_nums[0]); mth =
int(date_nums[1]); d = int(date_nums[2])
                                            hh = int(time_nums[0]); mm =
int(time_nums[1]); ss = int(time_nums[2])
                                            parsed_dt = _datetime_cls(y,
mth, d, hh, mm, ss)
                                            debug["parse_method"] =
"digits_to_ints"
                                        else:
                                            clean_date = "-
".join(date_nums[:3]) if date_nums else date_raw
```

```
                                                clean_time =
":".join(time_nums[:3]) if time_nums else time_raw
                                                try:
                                                    parsed_dt =
_datetime_cls.strptime(f"{clean_date} {clean_time}", "%Y-%m-%d %H:%M:%S")
                                                    debug["parse_method"] =
"strptime_fallback"
                                                except Exception as e_sp:

debug["parse_error_strptime_fallback"] = str(e_sp)
                                                    parsed_dt = None
                                            except Exception as e_par:
                                                debug["parse_error_digits"] =
repr(e_par)
                                                parsed_dt = None

                                            timestamp = parsed_dt
                                            iso = timestamp.isoformat() if
timestamp else None
                                            debug["timestamp_iso"] = iso

                                            # everything after matched time
                                            rest =
s_norm[dt_search.end():].lstrip(" :\t")
                                            # detect trailing username at very
end like "... : username"
                                            user_trail =
re.search(r'\s*:\s*(?P<user>[^\n:]{1,200})\s*$', rest)
                                            if user_trail:
                                                username =
user_trail.group("user").strip()
                                                text_part =
rest[:user_trail.start()].rstrip(" :\n\t")
                                            else:
                                                text_part = rest
                                            text = text_part.strip()
                                        else:
                                            # fallback: detect trailing username
only
                                            user_trail =
re.search(r'\s*:\s*(?P<user>[^\n:]{1,200})\s*$', s_norm)
                                            if user_trail:
                                                username =
user_trail.group("user").strip()
                                                text =
s_norm[:user_trail.start()].rstrip(" :\n\t").strip()
                                            else:
                                                text = s_norm.strip()
                                            debug["timestamp_iso"] = None

                                        # --- Ensure text is a single string and
wrap in triple single-quotes ---
                                        if text is None:
                                            text = ""
```

```
                                text_wrapped = "'''{}'''".format(text)

                                # --- Split extracted text on FIRST '?'
into question_asked and context_added ---
                                # operate on the unwrapped content (so
question_asked contains only the question text)
                                split_parts = re.split(r'\?', text,
maxsplit=1)
                                if len(split_parts) == 2:
                                    question_asked =
split_parts[0].strip() + '?'
                                    context_added =
split_parts[1].strip()
                                else:
                                    question_asked = text.strip()
                                    context_added = ""

                                debug["extracted_text_snippet"] =
repr(text_wrapped[:1000])
                                debug["extracted_username"] = username
                                debug["question_asked"] = question_asked
                                debug["context_added"] = context_added

                                return timestamp, iso, text_wrapped,
username, debug

                            try:
                                ts, iso, text, user, dbg =
extract_timestamp_text_username_robust(LLMPromptEdited,
fallback_username="ITF")

                                print("\n--- Parse Results ---")
                                print("timestamp:", ts)
                                print("iso:", iso)
                                print("username:", user)
                                print("text (first 400 chars):")
                                print(text[:400] + ("..." if len(text) >
400 else ""))
                                print("\n--- Debug ---")
                                import pprint
                                pprint.pprint(dbg)

                                Final_NewPromptEdited =
F"'message':{text} : 'score':None : 'score_conf':None : 'username':{user}
: 'timestamp':{ts}"

                            except:
                                Document_Prompt = f"{question_asked}
{context_added}"
                                print(f"\nDocument_Prompt :
{Document_Prompt}\n")
```

```
                                Final_NewPromptEdited =
F"'message':{Document_Prompt} : 'score':None : 'score_conf':None :
'username':{user} : 'timestamp':{ts}"

                            print('\r')
                            print("Final_NewPromptEdited : " +
Final_NewPromptEdited)
                            print('\n')

                            NewPromptEdited = Final_NewPromptEdited


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

##//////////////////////////////////////////////////////////////////////
//
    ##                RUN AND GO TO VLLM FOR VISION SYSTEM

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

                            START_VISION_KEYWORDS = [
                                "what is the",
                                "where is the",
                                "what do you",
                                "what is in",
                                "what is",
                                "what color",
                                "where do you",
                                "where will you"
                            ]

                            VISION_KEYWORDS = [
                                "do you see",
                                "the person",
                                "sight",
                                "the color",
                                "best way to go",
                                "observe",
                                "observing",
                                "spot",
                                "seeing",
                                "wearing",
                                "carrying",
                                "shoes",
                                "in this room",
                                "in the room",
                                "in the passage",
                                "showing",
                                "from here",
                                "this possition",
                                "this point",
                                "go next",
```

```
                                "go to next",
                                "the person pointing",
                                "that point"
                            ]

                            CODING_KEYWORD = [
                                "check",
                                "fix",
                                "python",
                                "code",
                                "a code",
                                "code for",
                                "arduino",
                                "c++",
                                "a python",
                                "in python",
                                "an arduino",
                                "in arduino ide",
                                "a c++",
                                "java",
                                "python code",
                                "arduino code",
                                "c++ code",
                                "javascript",
                                "typescript",
##                                "go",
##                                "rust",
                                "c#",
##                                "swift",
                                "kotlin",
                                "ruby",
                                "php",
                                "perl",
                                "matlab",
                                "scala",
                                "haskell",
                                "fortran",
                                "lua",
                                "dart",
                                "shell",
                                "bash",
                                "powershell"
                            ]

                            RAG_KEYWORDS = [
                                "rag", "this document", "this text",
"this pdf",
                                "this word", "this excel", "this story",
                                "this list", "this rag document", "this
rag text", "this rag pdf",
                                "this rag word", "this rag excel", "this
rag story",
                                "this rag list"
                            ]
```

```
                            print(f"[DEBUG MAIN JUST BEFORE LLM's]
final_updated_query : {final_updated_query}.")

                            if (any(word in final_updated_query for word
in START_VISION_KEYWORDS)
                                and any(word in final_updated_query for
word in VISION_KEYWORDS)
                                and not any(word in final_updated_query
for word in CODING_KEYWORD)
                            ):

                                print("[AI_VISION] Vision-related query
detected.")
                                NewPromptEditedSpeak =
NewPromptEditedSpeak.replace("telling you", "")
                                print('\r')
                                print("Diagnostic Testing.....Vision
Large Language Model")
                                print('\n')
                                print('\r')

                                NewPromptEdited =
NewPromptEdited.replace("+ username: Home","")

                                speech.AlfredSpeak(f"Let me see....")

                                VLLM_Prompt = NewPromptEdited
                                print(f"Let me tell you VLLM_Prompt
{VLLM_Prompt} ")

                                speech.AlfredSpeak(f"Let me tell you
{NewPromptEditedSpeak} ")

                                print(f"Let me tell you
{NewPromptEditedSpeak} ")

                                print("Going to Vision Large Language
Model.....!")

                                try :
                                    if MyToDoListEdited != []:
                                        MyToDoListEdited.pop(0)
                                        AlfredQueryOfflineNew =
MyToDoListEdited
                                        print("")
                                        print("AlfredQueryOffline New : "
+ str(AlfredQueryOfflineNew))
                                        print('')
                                except:
                                    pass


ai_vision.Vision_Model_Front_Look(VLLM_Prompt)
```

```
##########################################################################
##

##////////////////////////////////////////////////////////////////////////
/////
    ##                END OF VISION

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

                            else:


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

##////////////////////////////////////////////////////////////////////////
/////
    ##                RUN AND GO TO LLM FOR RAG INFORMATION

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

                                    if (not any(word in final_updated_query
for word in CODING_KEYWORD)
                                        and not any(word in
final_updated_query for word in VISION_KEYWORDS)
                                        and any(word in final_updated_query
for word in RAG_KEYWORDS)):

                                        print("Going to RAG LLM.....")
                                        speech.AlfredSpeak(f"Ahhh... Let me
memorize this and then {NewPromptEditedSpeak} ")
                                        print(f"NewPromptEdited :
{NewPromptEdited}")

ai_assistant.My_Ollama_LLM_RAG(NewPromptEdited, NewPromptEditedSpeak,
AlfredQueryOffline)


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

##////////////////////////////////////////////////////////////////////////
//
    ##                RUN AND GO TO LLM INFORMATION

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

                                    if (not any(word in final_updated_query
for word in CODING_KEYWORD)
```

```
                                    and not any(word in
final_updated_query for word in VISION_KEYWORDS)
                                    and not any(word in
final_updated_query for word in RAG_KEYWORDS)):

                                    print("Going to LLM.....")
                                    print(f"NewPromptEdited :
{NewPromptEdited}")
                                    NewPromptEditedSpeak =
NewPromptEditedSpeak.replace(" you ", "")

                                    speech.AlfredSpeak(f"Ahhh... Let me
think about {NewPromptEditedSpeak} ")

ai_assistant.My_Ollama_LLM(speech_sure_name, New_gender, NewPromptEdited,
NewPromptEditedSpeak, AlfredQueryOffline)



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

##//////////////////////////////////////////////////////////////////////
//
    ##                RUN AND GO TO LLM FOR CODING

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

                                elif (any(word in final_updated_query for
word in CODING_KEYWORD)
                                    and not any(word in
final_updated_query for word in VISION_KEYWORDS)
                                    and not any(word in
final_updated_query for word in RAG_KEYWORDS)):

                                    print("Going to Coder.....")
                                    speech.AlfredSpeak(f"Ahhh...
{NewPromptEditedSpeak} ")
                                    speech.AlfredSpeak(f"Fantastic, Let's
do some coding ...")
                                    print(f"NewPromptEdited :
{NewPromptEdited}")

ai_assistant.My_Ollama_LLM_Coder(NewPromptEdited, NewPromptEditedSpeak,
AlfredQueryOffline)

                except Exception as e:
                    print(e)

                print('return 2')
                Alfred_No_Ollama = 0
                return main(assistant, gui)
```

```
        except Exception as e:
            print(e)


        print('return 1')
        Alfred_No_Ollama = 0
        return main(assistant, gui)

if __name__ == "__main__":

    # Decide mode based on CLI flag
    web_mode = len(sys.argv) > 1 and sys.argv[1] in ("web", "--web")
    if web_mode:
        # ——— WebUI mode ———
        import webui
        print("□ Starting Alfred WebUI at http://localhost:5000 …")
        webui.run_webui()
        sys.exit(0)

    # ——— Desktop GUI mode ———
    print("□ Starting Alfred Desktop GUI …")

    # 1) Initialize the GUI
    gui = DesktopGUI()

    from assistant import assistant
    # 2) Wire the assistant & vision modules
    assistant.gui = gui
    ai_assistant = AI_Assistant(gui)
    ai_vision    = AI_VisionModule(gui)

    print("□ Starting Alfred Desktop GUI AUDIO and SPEECH…")
    # 3) Play startup sound & greeting
    playsound(Alfred_config.DRIVE_LETTER +
"Python_Env//New_Virtual_Env//Project_Files//My_Mp3//PTNK-on.mp3")
    assistant.wishme()
    time.sleep(1)

    import shutdown_helpers as sh

    vision_thread = threading.Thread(
        target=Vision_Who_InFront_Look_At_New,
        args=(get_current_speaker,),
        daemon=True,
        name="vision_tracker"
    )
    sh.register_thread(vision_thread, name="vision_tracker")
    vision_thread.start()

    main_thread = threading.Thread(
        target=main,
        args=(assistant, gui),
        daemon=True,
        name="main_loop"
```

```
)
sh.register_thread(main_thread, name="main_loop")
main_thread.start()

# 6) Run Tkinter mainloop
gui.run()
```