

EXPERIMENT-3

NAME: T KEERTHI PRIYA

ROLL NO:240701258

1. COMMAND LINE INTERFACE:

PYTHON CODE:

```
from typing import List, Dict

# --- Quiz data and shared logic ---

questions: List[Dict] = [

    {"q": "What is 2 + 2?", "a": "4"},

    {"q": "What is the capital of France?", "a": "Paris"},

    {"q": "What color do you get when you mix red and white?", "a": "pink"},

]


def run_quiz(prompt_func, output_func):

    """Run the quiz using provided prompt and output functions.

    prompt_func(question_text) -> user answer (string)
    output_func(text) -> display or speak text
    """

    score = 0

    total = len(questions)

    output_func(f"Welcome to the Simple Quiz! {total} questions.")

    for idx, item in enumerate(questions, start=1):

        q_text = f"Q{idx}. {item['q']}"

        answer = prompt_func(q_text)

        if answer is None:

            # Treat None as skipped

            output_func("No answer received; moving on.")
```

```
        continue

    if answer.strip().lower() == item['a'].strip().lower():

        output_func("Correct!")

        score += 1

    else:

        output_func(f"Incorrect. The correct answer is: {item['a']}")


output_func(f"Quiz complete. Your score: {score}/{total}")

return score


# --- CLI-specific prompt / output ---


def cli_prompt(question_text: str) -> str:

    try:

        return input(question_text + "\nYour answer: ")

    except (EOFError, KeyboardInterrupt):

        print() # newline

        return None


def cli_output(text: str):

    print(text)


if __name__ == "__main__":

    run_quiz(cli_prompt, cli_output)
```

```

PS C:\Users\Wthish\AppData\Local\Programs\Microsoft VS Code> & C:\Users\Wthish\AppData\Local\Programs\Python\Python313\python.exe /workspace/cli_quiz.py
Welcome to the Simple Quiz! 3 questions.
Q1. What is 2 + 2?
Your answer: 4
Correct!
Q2. What is the capital of France?
Your answer: paris
Correct!
Q3. What color do you get when you mix red and white?
Your answer: pink
Correct!
Quiz complete. Your score: 3/3
PS C:\Users\Wthish\AppData\Local\Programs\Microsoft VS Code>

```

2. GRAPHICAL USER INTERFACE:

PYTHON CODE:

```
import tkinter as tk
```

```
from tkinter import messagebox, ttk
```

```
from typing import List, Dict
```

```
# --- Quiz data and shared logic (same questions as other interfaces) ---
```

```
questions: List[Dict] = [
```

```
{
```

```
    "q": "What is 2 + 2?",
```

```
    "a": "4",
```

```
    "choices": ["3", "4", "5", "22"],
```

```
},
```

```
{
```

```
    "q": "What is the capital of France?",
```

```
    "a": "Paris",
```

```
    "choices": ["Berlin", "Madrid", "Paris", "Rome"],
```

```
},
```

```
{
```

```
    "q": "What color do you get when you mix red and white?",
```

```
    "a": "pink",
```

```
    "choices": ["Purple", "Pink", "Orange", "Brown"],  
    },  
]
```

```
def run_quiz(prompt_func, output_func):
```

```
    """Shared quiz runner used by other interfaces.
```

```
  
    It expects a `prompt_func` that returns the user's answer string  
    and an `output_func` to display feedback.
```

```
    """
```

```
    score = 0
```

```
    total = len(questions)
```

```
    output_func(f"Welcome to the Simple Quiz! {total} questions.")
```

```
    for idx, item in enumerate(questions, start=1):
```

```
        q_text = f"Q{idx}. {item['q']}"
```

```
        answer = prompt_func(q_text)
```

```
        if answer is None:
```

```
            output_func("No answer received; moving on.")
```

```
            continue
```

```
        if answer.strip().lower() == item['a'].strip().lower():
```

```
            output_func("Correct!")
```

```
            score += 1
```

```
        else:
```

```
            output_func(f"Incorrect. The correct answer is: {item['a']}")
```

```
    output_func(f"Quiz complete. Your score: {score}/{total}")
```

```
    return score
```

```
# --- GUI-specific code ---
```

```
class QuizGUI:
```

```

def __init__(self, master):

    self.master = master

    master.title("Simple Quiz")

    master.geometry("620x420")

    master.configure(bg="#f4f7fb")

    master.resizable(False, False)


# Styling

self.style = ttk.Style(master)

try:

    self.style.theme_use('clam')

except Exception:

    pass

self.style.configure('Card.TFrame', background='white')

self.style.configure('Title.TLabel', font=(None, 20, 'bold'), background='white')

self.style.configure('Question.TLabel', font=(None, 13), background='white')

self.style.configure('Choice.TButton', font=(None, 12), padding=10)

self.style.map('Choice.TButton', background=[('active', '#dfefff')])


self.index = 0

self.score = 0


# Outer padding frame

container = ttk.Frame(master, padding=20, style='Card.TFrame')

container.place(relx=0.5, rely=0.5, anchor=tk.CENTER, width=560, height=360)


# Title

self.title_label = ttk.Label(container, text='Simple Quiz', style='Title.TLabel')

self.title_label.pack(pady=(6, 10))


# Card for question and choices

```

```

card = ttk.Frame(container, padding=(12, 10), style='Card.TFrame')

card.pack(fill=tk.BOTH, expand=True)

# Question

self.question_var = tk.StringVar()

self.question_label = ttk.Label(card, textvariable=self.question_var, style='Question.TLabel', wraplength=500)

self.question_label.pack(pady=(4, 12))


# Choices grid - larger button appearance

self.choices_frame = ttk.Frame(card, style='Card.TFrame')

self.choices_frame.pack(fill=tk.BOTH, expand=True)


self.choice_buttons = []

for r in range(2):

    for c in range(2):

        i = r * 2 + c

        btn = ttk.Button(self.choices_frame, text=f'Choice {i+1}', style='Choice.TButton', command=lambda i=i:
self.on_choice(i))

        btn.grid(row=r, column=c, padx=8, pady=8, sticky=tk.NSEW)

        self.choice_buttons.append(btn)

for i in range(2):

    self.choices_frame.grid_columnconfigure(i, weight=1)


# Feedback and progress

bottom = ttk.Frame(container, padding=(6, 6), style='Card.TFrame')

bottom.pack(fill=tk.X)


self.feedback_var = tk.StringVar()

self.feedback_label = ttk.Label(bottom, textvariable=self.feedback_var, foreground='#2b7a78', background='white')

self.feedback_label.pack(side=tk.LEFT)


self.progress = ttk.Progressbar(bottom, maximum=len(questions), length=220)

```

```
self.progress.pack(side=tk.RIGHT)
```

```
# Controls
```

```
controls = ttk.Frame(container, style='Card.TFrame')
```

```
controls.pack(pady=(8, 0))
```

```
self.next_button = ttk.Button(controls, text='Next', command=self.next_question, state=tk.DISABLED)
```

```
self.next_button.grid(row=0, column=0, padx=6)
```

```
self.restart_button = ttk.Button(controls, text='Restart', command=self.restart)
```

```
self.restart_button.grid(row=0, column=1, padx=6)
```

```
# Keyboard bindings for choices 1-4 and Enter for Next
```

```
master.bind('1', lambda e: self._kbd_choice(0))
```

```
master.bind('2', lambda e: self._kbd_choice(1))
```

```
master.bind('3', lambda e: self._kbd_choice(2))
```

```
master.bind('4', lambda e: self._kbd_choice(3))
```

```
master.bind('<Return>', lambda e: self._kbd_next())
```

```
self.show_question()
```

```
def _kbd_choice(self, idx):
```

```
    if idx < len(self.choice_buttons) and self.choice_buttons[idx]['state'] == tk.NORMAL:
```

```
        self.on_choice(idx)
```

```
def _kbd_next(self):
```

```
    if self.next_button['state'] == tk.NORMAL:
```

```
        self.next_question()
```

```
def show_question(self):
```

```
    if self.index >= len(questions):
```

```
self.finish_quiz()
```

```
return
```

```
q = questions[self.index]
```

```
self.question_var.set(f"Q{self.index+1}. {q['q']}")
```

```
choices = q.get('choices') or [q['a']]
```

```
while len(choices) < 4:
```

```
    choices.append(choices[-1])
```

```
for btn, text in zip(self.choice_buttons, choices[:4]):
```

```
    btn.config(text=text, state=tk.NORMAL)
```

```
self.feedback_var.set(f"Question {self.index+1} of {len(questions)}")
```

```
self.progress['value'] = self.index
```

```
self.next_button.config(state=tk.DISABLED)
```

```
def on_choice(self, choice_index: int):
```

```
    q = questions[self.index]
```

```
    selected_text = self.choice_buttons[choice_index].cget('text')
```

```
    correct = q['a']
```

```
    for btn in self.choice_buttons:
```

```
        btn.config(state=tk.DISABLED)
```

```
    if selected_text.strip().lower() == correct.strip().lower():
```

```
        self.score += 1
```

```
        self.feedback_var.set('Correct )
```

```
    else:
```

```
        self.feedback_var.set(f"Incorrect  (Correct: {correct})")
```

```

self.next_button.config(state=tk.NORMAL)

def next_question(self):

    self.index += 1

    self.show_question()

def finish_quiz(self):

    self.progress['value'] = len(questions)

    # Summary window
    summary = tk.Toplevel(self.master)

    summary.title('Results')

    summary.geometry('360x180')

    summary.transient(self.master)

    summary.grab_set()

    ttk.Label(summary, text='Quiz Complete', font=(None, 16, 'bold')).pack(pady=(12, 6))

    ttk.Label(summary, text=f'Your score: {self.score}/{len(questions)}', font=(None, 13)).pack(pady=(0, 10))

    btn_frame = ttk.Frame(summary)

    btn_frame.pack(pady=8)

    ttk.Button(btn_frame, text='Restart', command=lambda: (summary.destroy(), self.restart())).grid(row=0, column=0,
    padx=8)

    ttk.Button(btn_frame, text='Close', command=self.master.quit).grid(row=0, column=1, padx=8)

def restart(self):

    self.index = 0

    self.score = 0

    self.show_question()

if __name__ == "__main__":

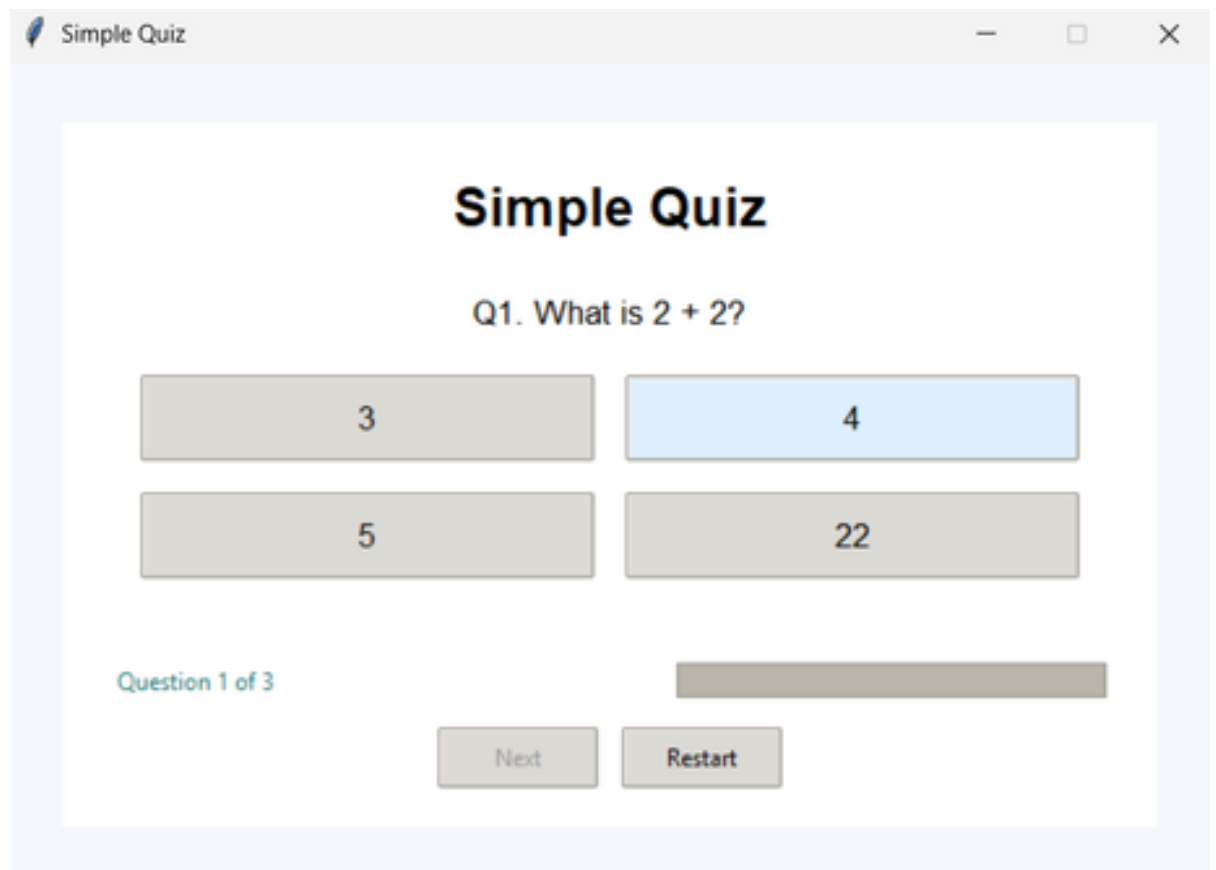
    root = tk.Tk()

    root.geometry("480x320")

```

```
app = QuizGUI(root)
```

```
root.mainloop()
```



3. VOICE USER INTERFACE:

PYTHON CODE:

```
import sys
```

```
import threading
```

```
import time
```

```
from queue import Queue
```

```
# Optional voice libs: do not raise if missing — fall back to typed I/O
```

```
try:
```

```
    import speech_recognition as sr
```

```
    _HAS_STT = True
```

except Exception:

sr = None

_HAS_STT = False

try:

import pyttsx3

_HAS_TTS = True

except Exception:

pyttsx3 = None

_HAS_TTS = False

def speak(text):

"""Speak using pyttsx3 if available, otherwise print."""

if _HAS_TTS and pyttsx3 is not None:

try:

engine = pyttsx3.init()

engine.say(text)

engine.runAndWait()

return

except Exception:

pass

print(text)

def listen_once(recognizer, mic, seconds=3):

"""Try to listen from mic and return recognized text; fallback to typed input."""

if _HAS_STT and sr is not None and recognizer is not None and mic is not None:

print(f"Speak now (listening for up to {seconds} seconds)...")

try:

audio = recognizer.listen(mic, phrase_time_limit=seconds)

except Exception:

print('No speech detected (timeout or mic error).')

```

    return None

try:

    return recognizer.recognize_google(audio)

except sr.UnknownValueError:

    print('Could not understand audio.')

except Exception as e:

    print('Recognition error:', e)

    return None

# Fallback: typed input

try:

    typed = input(f"(type) Your answer (or leave blank): ")

    return typed if typed.strip() else None

except (EOFError, KeyboardInterrupt):

    print()

    return None


def interactive_loop():

    """Interactive terminal loop — uses voice if available, otherwise typed input."""

    recognizer = None

    microphone = None

    if _HAS_STT and sr is not None:

        try:

            recognizer = sr.Recognizer()

            microphone = sr.Microphone()

        except Exception:

            recognizer = None

            microphone = None

    history = []

    timer = 3

```

```
print("\nTerminal VUI demo — voice optional")
```

```
print("Commands: Enter=SpeakNow, 't'=set timer, 'h'=history, 's'=speak text, 'q'=quit, 'quiz'=run quiz")
```

```
while True:
```

```
    try:
```

```
        cmd = input('\n> ').strip()
```

```
    except (KeyboardInterrupt, EOFError):
```

```
        print('\nExiting.')
```

```
        break
```

```
if cmd == '':
```

```
    # Speak now
```

```
    if recognizer and microphone:
```

```
        with microphone as source:
```

```
            try:
```

```
                recognizer.adjust_for_ambient_noise(source, duration=0.4)
```

```
            except Exception:
```

```
                pass
```

```
            text = listen_once(recognizer, source, seconds=timer)
```

```
    else:
```

```
        text = listen_once(None, None, seconds=timer)
```

```
    if text:
```

```
        print('You said:', text)
```

```
        history.append(text)
```

```
        handle_spoken(text)
```

```
elif cmd == 't':
```

```
    val = input('Set timer seconds (e.g. 3): ').strip()
```

```
    try:
```

```
        v = int(val)
```

```
if v <= 0:
```

```
    raise ValueError
```

```
timer = v
```

```
print('Timer set to', timer, 'seconds')
```

```
except Exception:
```

```
    print('Invalid number.')
```

```
elif cmd == 'h':
```

```
    print('\nTranscript history:')
```

```
    for i, h in enumerate(history[-20:], start=1):
```

```
        print(f' {i}.', h)
```

```
elif cmd == 's':
```

```
    text = input('Text to speak: ').strip()
```

```
    if text:
```

```
        speak(text)
```

```
elif cmd == 'quiz':
```

```
    vui = VUI()
```

```
    def vui_prompt(q_text: str) -> str:
```

```
        return vui.listen(q_text)
```

```
    def vui_output(text: str):
```

```
        vui.speak(text)
```

```
        time.sleep(0.3)
```

```
    run_quiz(vui_prompt, vui_output)
```

```
elif cmd == 'q':
```

```
    print('Quitting.')
```

```
    break
```

```
else:
```

```

print("Unknown command. Press Enter to 'Speak Now', or 'q' to quit.")

if __name__ == '__main__':
    interactive_loop()

from typing import List, Dict

import time

# Try to import optional voice libraries
try:
    import pyttsx3
    _HAS_TTS = True
except Exception:
    pyttsx3 = None
    _HAS_TTS = False

try:
    import speech_recognition as sr
    _HAS_STT = True
except Exception:
    sr = None
    _HAS_STT = False

# --- Quiz data and shared logic ---
questions: List[Dict] = [
    {"q": "What is 2 + 2?", "a": "4"},
    {"q": "What is the capital of France?", "a": "Paris"},
    {"q": "What color do you get when you mix red and white?", "a": "pink"},
]

```

```

def run_quiz(prompt_func, output_func):

```

```

score = 0

total = len(questions)

output_func(f"Welcome to the Simple Quiz! {total} questions.")


for idx, item in enumerate(questions, start=1):

    q_text = f"Q{idx}. {item['q']}"

    answer = prompt_func(q_text)

    if answer is None:

        output_func("No answer received; moving on.")

        continue

    if answer.strip().lower() == item['a'].strip().lower():

        output_func("Correct!")

        score += 1

    else:

        output_func(f"Incorrect. The correct answer is: {item['a']}")


output_func(f"Quiz complete. Your score: {score}/{total}")

return score

```

--- VUI helpers ---

```
class VUI:
```

```

    def __init__(self):

        self.tts_engine = None

        if _HAS_TTS:

            try:

                self.tts_engine = pyttsx3.init()

            except Exception:

                self.tts_engine = None


        if _HAS_STT:

            try:

```

```

        self.recognizer = sr.Recognizer()

        self.microphone = sr.Microphone()

    except Exception:

        self.recognizer = None

        self.microphone = None

    else:

        self.recognizer = None

        self.microphone = None

def speak(self, text: str):

    """Speak the text if TTS is available, otherwise print it."""

    if self.tts_engine:

        try:

            self.tts_engine.say(text)

            self.tts_engine.runAndWait()

        except Exception:

            print("[TTS failed]", text)

    else:

        print(text)

def listen(self, prompt: str, timeout: int = 5) -> str:

    """Try to listen from the microphone and return recognized text.

    If speech libraries are not available or recognition fails, fall back to typed input.

    """

    # Ask the user first (via voice or print)

    self.speak(prompt)

    # If speech recognition is configured, attempt to use it

    if self.recognizer and self.microphone:

        try:

            with self.microphone as source:

```

```
self.recognizer.adjust_for_ambient_noise(source, duration=0.5)
```

```
self.speak("Please speak your answer now.")
```

```
audio = self.recognizer.listen(source, timeout=timeout)
```

```
# Use recognizer's default (Google Web Speech) recognizer if available
```

```
try:
```

```
    text = self.recognizer.recognize_google(audio)
```

```
    return text
```

```
except Exception:
```

```
    self.speak("Sorry, I didn't catch that.")
```

```
    return None
```

```
except Exception:
```

```
    # Any microphone/recognition error -> fallback
```

```
    self.speak("Voice input is not available; please type your answer.")
```

```
# Fallback: typed input
```

```
try:
```

```
    # Print prompt and accept typed answer
```

```
    typed = input(prompt + "\nYour answer (type): ")
```

```
    return typed
```

```
except (EOFError, KeyboardInterrupt):
```

```
    print()
```

```
    return None
```

```
if __name__ == "__main__":
```

```
    vui = VUI()
```

```
def vui_prompt(q_text: str) -> str:
```

```
    return vui.listen(q_text)
```

```
def vui_output(text: str):
```

```
    vui.speak(text)
```

```
# small pause so speech is easier to follow
```

```
time.sleep(0.3)
```

```
# Explain capabilities to the user
```

```
if vui.tts_engine is None and (vui.recognizer is None or vui.microphone is None):
```

```
    print("Voice libraries not available. Running in text fallback mode.")
```

```
    print("To add voice support: pip install pyttsx3 SpeechRecognition pyaudio")
```

```
run_quiz(vui_prompt, vui_output)
```

```
Starting quiz: 3 questions. Press Enter to Speak Now for each question.
```

```
Q1. What is 2 + 2?
```

```
Press Enter to Speak Now...
```

```
(you press Enter, then speak "4" or type "4")
```

```
Recognized: 4
```

```
Correct!
```

```
Q2. What is the capital of France?
```

```
Press Enter to Speak Now...
```

```
(you press Enter, then speak "Paris" or type "Paris")
```

```
Recognized: Paris
```

```
Correct!
```

```
Q3. What color do you get when you mix red and white?
```

```
Press Enter to Speak Now...
```

```
(you press Enter, then speak "pink" or type "pink")
```

```
Recognized: pink
```

```
Correct!
```

```
Quiz complete. Your score: 3/3
```