Task Notifier:

Python Program:

```
import tkinter as tk
from tkinter import ttk, messagebox, filedialog, image_names
import json
from datetime import datetime
import time
import threading
import os
import base64
import tempfile
from PIL import Image
import winsound  # For Windows sound
import platform  # To check operating system
# Try importing different notification libraries with fallbacks
try:
    from win10toast import ToastNotifier
    from winotify import Notification, audio
    WINDOWS_NOTIFICATIONS_AVAILABLE = True
except ImportError:
    WINDOWS_NOTIFICATIONS_AVAILABLE = False

try:
    from plyer import notification as plyer_notification
    PLYER_AVAILABLE = True
except ImportError:
    PLYER_AVAILABLE = False

class NotifierApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Desktop Notifier")
        self.root.geometry("600x400")
```

```
# Set default icon path
self.icon_path = None
self.default_icon = "A:\Study-Store\Python\Icon.ico"

# Add image path variable
self.image_path = None
self.default_image = os.path.join("A:\Study-Store\Python\Icon.ico")  # Default notification image

# Set default sound path
self.sound_path = None
self.default_sound = "A:\Study-Store\Python\notification.wav"  # Default sound file

current_dir = os.path.dirname(os.path.abspath(__file__))
self.icon_path = None
self.default_icon = os.path.join(current_dir, "A:\Study-Store\Python\Icon.ico")
# Initialize Windows notification if available
if os.name == 'nt' and WINDOWS_NOTIFICATIONS_AVAILABLE:
    try:
        self.toaster = ToastNotifier()
    except Exception:
        self.toaster = None
# Verify icon exists and is accessible
if not os.path.exists(self.default_icon):
    messagebox.showwarning("Warning", f"Default icon not found at: {self.default_icon}")

# Try to set window icon
try:
    self.root.iconbitmap(self.default_icon)
except tk.TclError:
    pass

# Load existing notifications from file
self.notifications = self.load_notifications()

# Create main frame
```

```
        self.main_frame = ttk.Frame(root, padding="10")
        self.main_frame.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))

        # Create notification form
        self.create_form()

        # Create notification list
        self.create_list()

        # Start notification checker thread
        self.checker_thread = threading.Thread(target=self.check_notifications, daemon=True)
        self.checker_thread.start()

    def choose_sound(self):
        """Function to choose notification sound"""
        filetypes = [
            ('WAV files', '*.wav'),
            ('All files', '*.*')
        ]
        sound_path = filedialog.askopenfilename(
            title="Choose notification sound",
            filetypes=filetypes
        )

        if sound_path:
            self.sound_path = sound_path
            self.sound_label.config(text=os.path.basename(sound_path))

    def test_sound(self):
        """Function to test the selected sound"""
        try:
            if platform.system() == 'Windows':
                sound_file = self.sound_path if self.sound_path else self.default_sound
                winsound.PlaySound(sound_file, winsound.SND_FILENAME)
            else:
                messagebox.showinfo("Info", "Sound testing is only available on Windows")
        except Exception as e:
            messagebox.showerror("Error", f"Failed to play sound: {str(e)}")

    def play_notification_sound(self):
        """Function to play notification sound"""
        try:
            if platform.system() == 'Windows':
                sound_file = self.sound_path if self.sound_path else self.default_sound
                winsound.PlaySound(sound_file, winsound.SND_FILENAME | winsound.SND_ASYNC)
        except Exception as e:
            print(f"Failed to play sound: {str(e)}")

    def create_form(self):
        # Form frame
        form_frame = ttk.LabelFrame(self.main_frame, text="Create Notification", padding="10")
        form_frame.grid(row=0, column=0, padx=5, pady=5, sticky=(tk.W, tk.E))

        # Title
        ttk.Label(form_frame, text="Title:").grid(row=0, column=0, sticky=tk.W)
        self.title_var = tk.StringVar()
        ttk.Entry(form_frame, textvariable=self.title_var).grid(row=0, column=1, sticky=(tk.W, tk.E))

        # Image selection
        #ttk.Label(form_frame, text="Image:").grid(row=3, column=0, sticky=tk.W)
        #self.image_label = ttk.Label(form_frame, text="No image selected")
        #self.image_label.grid(row=3, column=1, sticky=tk.W)
        #ttk.Button(form_frame, text="Choose Image", command=self.choose_image).grid(row=3, column=2, padx=5)

        # Message
        ttk.Label(form_frame, text="Message:").grid(row=1, column=0, sticky=tk.W)
        self.message_var = tk.StringVar()
        ttk.Entry(form_frame, textvariable=self.message_var).grid(row=1, column=1, sticky=(tk.W, tk.E))

        # Time
        ttk.Label(form_frame, text="Time (HH:MM):").grid(row=2, column=0, sticky=tk.W)
        self.time_var = tk.StringVar()
        ttk.Entry(form_frame, textvariable=self.time_var).grid(row=2, column=1, sticky=(tk.W, tk.E))

        # Buttons
        btn_frame = ttk.Frame(form_frame)
        btn_frame.grid(row=4, column=0, columnspan=3, pady=10)

        ttk.Button(btn_frame, text="Create", command=self.create_notification).grid(row=0, column=0, padx=5)
        ttk.Button(btn_frame, text="Update", command=self.update_notification).grid(row=0, column=1, padx=5)
        ttk.Button(btn_frame, text="Delete", command=self.delete_notification).grid(row=0, column=2, padx=5)

    def choose_icon(self):
        filetypes = [
            ('ICO files', '*.ico'),
            ('PNG files', '*.png'),
            ('All files', '*.*')
        ]
        icon_path = filedialog.askopenfilename(
            title="Choose notification icon",
            filetypes=filetypes
        )

        if icon_path:
            self.icon_path = icon_path
            self.icon_label.config(text=os.path.basename(icon_path))

            # Convert icon to ICO if it's a PNG
            if icon_path.lower().endswith('.png'):
                try:
                    img = Image.open(image_names())
                    with tempfile.NamedTemporaryFile(delete=False, suffix='.ico') as tmp_file:
                        img.save(tmp_file.name, format='ICO')
                        self.image_path = tmp_file.name
                except Exception as e:
```

```
            messagebox.showerror("Error", f"Failed to convert image: {str(e)}")
            self.image_path = None
            self.image_label.config(text="No image selected")

    def create_list(self):
        # List frame
        list_frame = ttk.LabelFrame(self.main_frame, text="Notifications", padding="10")
        list_frame.grid(row=1, column=0, padx=5, pady=5, sticky=(tk.W, tk.E))

        # Treeview
        columns = ('title', 'message', 'time', 'image')
        self.tree = ttk.Treeview(list_frame, columns=columns, show='headings')

        # Define headings
        self.tree.heading('title', text='Title')
        self.tree.heading('message', text='Message')
        self.tree.heading('time', text='Time')
        self.tree.heading('image', text='image')

        # Column widths
        self.tree.column('title', width=150)
        self.tree.column('message', width=250)
        self.tree.column('time', width=100)
        self.tree.column('image', width=100)

        self.tree.grid(row=0, column=0, sticky=(tk.W, tk.E))

        # Bind selection event
        self.tree.bind('<<TreeviewSelect>>', self.item_selected)

        # Load existing notifications
        self.refresh_list()

    def choose_image(self):
        """Function to choose notification image"""
```

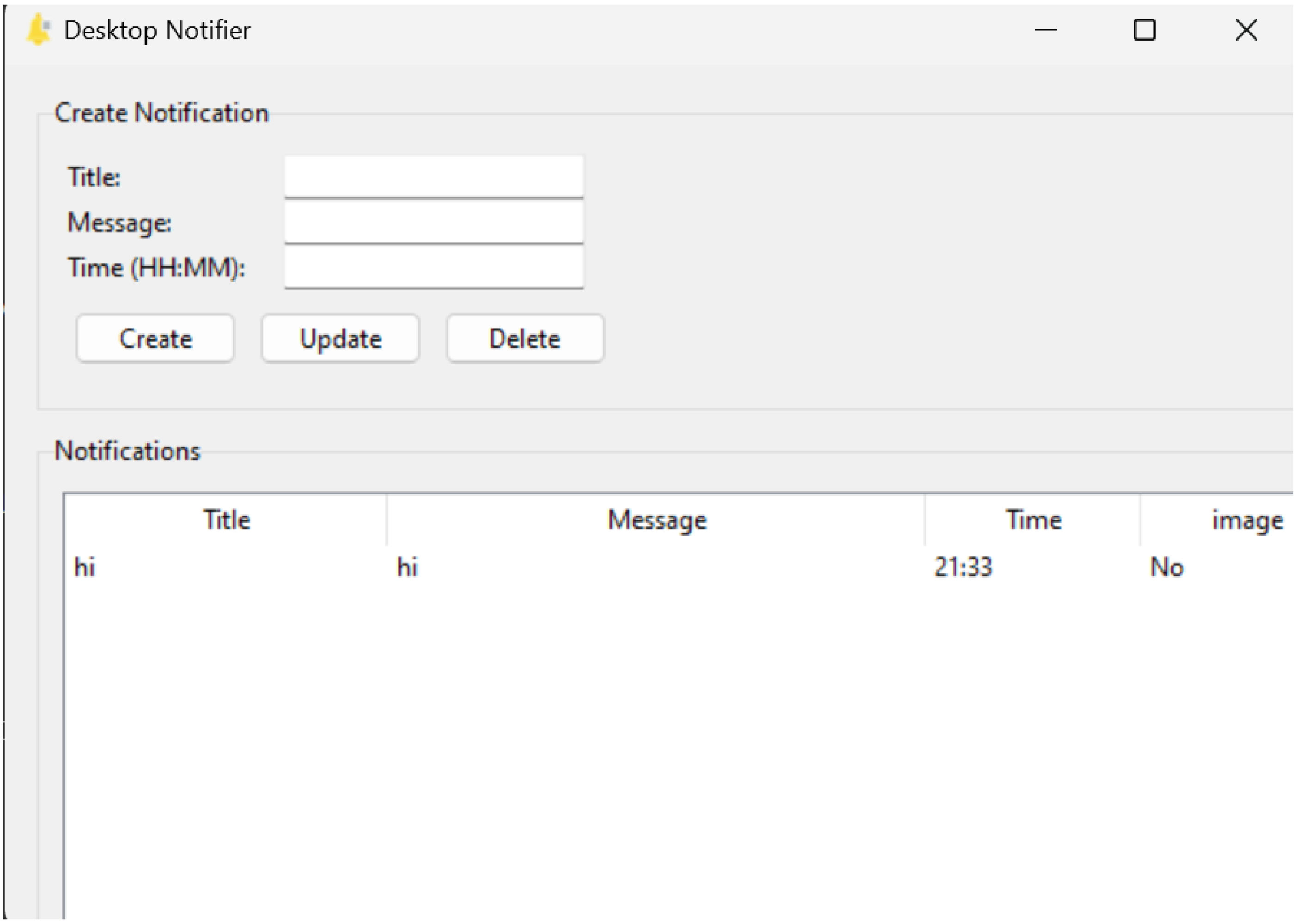
```
filetypes = [
    ('Image files', '*.png *.jpg *.jpeg *.bmp *.ico *.gif'),
    ('All files', '*.*')
]
image_path = filedialog.askopenfilename(
    title="Choose notification image",
    filetypes=filetypes
)

if image_path:
    try:
        # Open and validate the image
        img = Image.open(image_path)

        # Create temporary directory if it doesn't exist
        temp_dir = os.path.join(os.path.dirname(os.path.abspath(__file__)), 'temp')
        os.makedirs(temp_dir, exist_ok=True)

        # Convert to PNG and resize if necessary
        # Windows notifications work best with images around 364x180 pixels
        img = img.convert('RGBA')
        aspect_ratio = img.width / img.height
        if aspect_ratio > 2:  # wider than 2:1
            new_width = 364
            new_height = int(new_width / aspect_ratio)
        else:
            new_height = 180
            new_width = int(new_height * aspect_ratio)
        img = img.resize((new_width, new_height), Image.Resampling.LANCZOS)

        # Save as temporary PNG file
        temp_image_path = os.path.join(temp_dir, 'temp_notification.png')
        img.save(temp_image_path, 'PNG')

        self.image_path = temp_image_path
```

```
            self.image_label.config(text=os.path.basename(image_path))
        except Exception as e:
            messagebox.showerror("Error", f"Failed to process image: {str(e)}")
            self.image_path = None
            self.image_label.config(text="No image selected")

    def create_notification(self):
        title = self.title_var.get().strip()
        message = self.message_var.get().strip()
        time_str = self.time_var.get().strip()

        if not all([title, message, time_str]):
            messagebox.showerror("Error", "All fields are required!")
            return

        try:
            datetime.strptime(time_str, "%H:%M")
        except ValueError:
            messagebox.showerror("Error", "Invalid time format! Use HH:MM")
            return

        image_data = None
        if self.image_path and os.path.exists(self.image_path):
            try:
                with open(self.image_path, 'rb') as img_file:
                    image_data = base64.b64encode(img_file.read()).decode('utf-8')
            except Exception as e:
                messagebox.showwarning("Warning", f"Failed to load image: {str(e)}")

        # If icon is selected, encode it to base64
        image_data = None
        if self.image_path and os.path.exists(self.image_path):
            try:
                img = Image.open(self.image_path)
                # Save as ICO if it's not already
                if not self.image_path.lower().endswith('.ico'):
                    with tempfile.NamedTemporaryFile(delete=False, suffix='.ico') as tmp_file:
                        img.save(tmp_file.name, format='ICO')
                        self.image_path = tmp_file.name
                with open(self.image_path, 'rb') as image_data:
                    image_data = base64.b64encode(image_data.read()).decode('utf-8')
            except Exception as e:
                messagebox.showwarning("Warning", f"Failed to load image: {str(e)}")
                self.image_path = None
        notification_data = {
            "title": title,
            "message": message,
            "time": time_str,
            "image": image_data
        }

        self.notifications.append(notification_data)
        self.save_notifications()
        self.refresh_list()
        self.clear_form()

    def send_notification(self, title, message, image_path=None, sound_path=None):
        if os.name == 'nt' and WINDOWS_NOTIFICATIONS_AVAILABLE:
            try:
                # Try using winotify first as it has better image support
                toast = Notification(
                    app_id="NotifierApp",
                    title=title,
                    msg=message,
                    icon=self.default_icon,  # Use icon for the app icon
                    duration="long"
                )

                # Add image if available
                if image_path and os.path.exists(image_path):
```

```
            toast.add_icon(image_path)

        if sound_path:
            toast.set_audio(audio.Default, loop=False)

        toast.show()
    except Exception as e:
        print(f"Winotify error: {str(e)}")
        # Fallback to win10toast (note: won't show image)
        try:
            if hasattr(self, 'toaster') and self.toaster:
                self.toaster.show_toast(
                    title=title,
                    msg=message,
                    image_path=self.default_image,
                    duration=10,
                    threaded=True
                )
        except Exception as e:
            print(f"Win10toast error: {str(e)}")
            messagebox.showerror("Error", f"Failed to send notification: {str(e)}")

        #set custom sound if provided
        if sound_path:
           toast.set_audio(audio.Default,loop=False)
        toast.show()
    except Exception as e:
        messagebox.showerror("Error",f"Failed to send notification: {str(e)}")

    except Exception:
        try:
            toast = Notification(
                app_id="NotifierApp",
                title=title,
                msg=message,
                        image=image_path or self.default_image,
                        duration="long"
                    )
                    toast.show()
                except Exception as e:
                    messagebox.showerror("Error", f"Failed to send notification: {str(e)}")
        elif PLYER_AVAILABLE:
            try:
                plyer_notification.notify(
                    title=title,
                    message=message,
                    app_image=image_path or self.default_image,
                    timeout=10
                )
            except Exception as e:
                messagebox.showerror("Error", f"Failed to send notification: {str(e)}")
        else:
            messagebox.showwarning("Warning", "No notification system available")

    def check_notifications(self):
        while True:
            current_time = datetime.now().strftime("%H:%M")
            for notif in self.notifications:
                if notif["time"] == current_time:
                    image_path = None
                    if notif.get("image"):
                        try:
                            # Create temp directory if it doesn't exist
                            temp_dir = os.path.join(os.path.dirname(os.path.abspath(__file__)), 'temp')
                            os.makedirs(temp_dir, exist_ok=True)

                            # Save the image data to a temporary file
                            image_data = base64.b64decode(notif["image"])
                            temp_image_path = os.path.join(temp_dir, f'notification_image_{int(time.time())}.png')
                            with open(temp_image_path, 'wb') as img_file:
                                img_file.write(image_data)
                            image_path = temp_image_path
                        except Exception as e:
                            print(f"Failed to process notification image: {str(e)}")
                            image_path = None

                    self.send_notification(
                        notif["title"],
                        notif["message"],
                        image_path,
                        notif.get("sound", self.default_sound)
                    )
                    self.send_notification(notif["title"], notif["message"], image_path)

                    # Clean up temporary image file
                    if image_path and os.path.exists(image_path):
                        try:
                            os.unlink(image_path)
                        except Exception as e:
                            print(f"Failed to clean up temporary image: {str(e)}")

            time.sleep(30)  # Check every 30 seconds

    def delete_notification(self):
        selected = self.tree.selection()
        if not selected:
            messagebox.showwarning("Warning", "Please select a notification to delete!")
            return

        confirm = messagebox.askyesno("Confirm Delete", "Are you sure you want to delete this notification?")
        if not confirm:
            return

        index = self.tree.index(selected[0])
        self.notifications.pop(index)
        self.save_notifications()
        self.refresh_list()
        self.clear_form()

    def update_notification(self):
        selected = self.tree.selection()
        if not selected:
            messagebox.showwarning("Warning", "Please select a notification to update!")
            return

        index = self.tree.index(selected[0])
        title = self.title_var.get().strip()
        message = self.message_var.get().strip()
        time_str = self.time_var.get().strip()

        if not all([title, message, time_str]):
            messagebox.showerror("Error", "All fields are required!")
            return

        try:
            datetime.strptime(time_str, "%H:%M")
        except ValueError:
            messagebox.showerror("Error", "Invalid time format! Use HH:MM")
            return

        image_data = None
        if self.image_path and os.path.exists(self.image_path):
            try:
                with open(self.image_path, 'rb') as image_file:
                    image_data = base64.b64encode(image_file.read()).decode('utf-8')
            except Exception as e:
                messagebox.showwarning("Warning", f"Failed to load icon: {str(e)}")

        self.notifications[index] = {
            "title": title,
        "message": message,
        "time": time_str,
        "image": image_data
    }

    self.save_notifications()
    self.refresh_list()
    self.clear_form()

def item_selected(self, event):
    selected = self.tree.selection()
    if selected:
        index = self.tree.index(selected[0])
        notification = self.notifications[index]
        self.title_var.set(notification["title"])
        self.message_var.set(notification["message"])
        self.time_var.set(notification["time"])

        if notification.get("image"):
            try:
                image_data = base64.b64decode(notification["image"])
                with tempfile.NamedTemporaryFile(delete=False, suffix='.ico') as tmp_file:
                    tmp_file.write(image_data)
                    self.image_path = tmp_file.name
                    self.image_label.config(text="Saved image")
            except Exception:
                self.image_path = None
                self.image_label.config(text="No image selected")
        else:
            self.image_path = None
            self.image_label.config(text="No image selected")

def clear_form(self):
    self.title_var.set("")
    self.message_var.set("")
```

```
        self.time_var.set("")
        self.image_path = None
        self.image_label.config(text="No image selected")

    def refresh_list(self):
        for item in self.tree.get_children():
            self.tree.delete(item)

        for notification in self.notifications:
            self.tree.insert('', tk.END, values=(
                notification["title"],
                notification["message"],
                notification["time"],
                "Yes" if notification.get("image") else "No"
            ))

    def load_notifications(self):
        try:
            if os.path.exists("notifications.json"):
                with open("notifications.json", "r") as f:
                    return json.load(f)
        except Exception as e:
            messagebox.showwarning("Warning", f"Failed to load notifications: {str(e)}")
        return []

    def save_notifications(self):
        try:
            with open("notifications.json", "w") as f:
                json.dump(self.notifications, f)
                print(self.notifications)
        except Exception as e:
            messagebox.showerror("Error", f"Failed to save notifications: {str(e)}")

def main():
    root = tk.Tk()
    NotifierApp(root)
    root.mainloop()


if __name__ == "__main__":
    main()
```

Result:



Title	Message	Time	image
hi	hi	21:33	No