**MINI PROJECT 2**

**BATCH-06**

**FILE MANAGEMENT SYSTEM**

**EXECUTION CODE IN PYTHON**

From tkinter import \*

Import tkinter as tk

Import tkinter.filedialog as fd

From tkinter import filedialog, messagebox

Import tkinter.messagebox as mb

Import os

Import shutil

From datetime import datetime

Def open\_file():

   file = fd.askopenfilename(title='choose a file of any type', filetypes=[("all files", "\*.\*")])

   os.startfile(os.path.abspath(file))

Def copy\_file():

   file\_to\_copy = fd.askopenfilename(title='choose a file to copy', filetypes=[("all files", "\*.\*")])

   dir\_to\_copy\_to = fd.askdirectory(title="in which folder to copy to?")

   try:

       shutil.copy(file\_to\_copy, dir\_to\_copy\_to)

       mb.showinfo(title='file copied!', message='your desired file has been copied to your desired location')

   except:

       mb.showerror(title='error!', message='we were unable to copy your file to the desired location. Please try again')

Def delete\_file():

   file = fd.askopenfilename(title='choose a file to delete', filetypes=[("all files", "\*.\*")])

   os.remove(os.path.abspath(file))

   mb.showinfo(title='file deleted', message='your desired file has been deleted')

Def rename\_file():

   file = fd.askopenfilename(title='choose a file to rename', filetypes=[("all files", "\*.\*")])

   rename\_wn = toplevel(root)

   rename\_wn.title("rename the file to")

   rename\_wn.geometry("250x70"); rename\_wn.resizable(0, 0)

   label(rename\_wn, text='what should be the new name of the file?', font=("times new roman", 10)).place(x=0, y=0)

   new\_name = entry(rename\_wn, width=40, font=("times new roman", 10))

   new\_name.place(x=0, y=30)   #dialogue box

   new\_file\_name = os.path.join(os.path.dirname(file), new\_name.get()+os.path.splitext(file)[1])

   os.rename(file, new\_file\_name)

   mb.showinfo(title="file renamed", message='your desired file has been renamed')

Def sort\_files():

    src\_folder = filedialog.askdirectory(title="select the source folder")

    dst\_folder = filedialog.askdirectory(title="select the destination folder")

    for filename in os.listdir(src\_folder):

        file\_path = os.path.join(src\_folder, filename)

        if os.path.isfile(file\_path):

            extension = os.path.splitext(filename)[1][1:]

            if extension:

                extension\_folder = os.path.join(dst\_folder, extension)

                if not os.path.exists(extension\_folder):

                    os.makedirs(extension\_folder)

                shutil.move(file\_path, os.path.join(extension\_folder, filename))

    root = tk.tk()

    root.withdraw()

    tk.messagebox.showinfo(title="done", message="files sorted successfully.")

Def merge\_files():

    files=filedialog.askopenfilenames(title="select files to merge");

    dst\_file = filedialog.asksaveasfilename(title="save merged file as", defaultextension=".merged")

    print(files)

    with open(dst\_file, 'wb') as f:

        for file\_path in files:

            with open(file\_path, "rb") as src\_f:

                f.write(src\_f.read())

    root = tk.tk()

    root.withdraw()

    tk.messagebox.showinfo(title="done", message="files merged successfully.")

# def merge\_files():

#     file1 = filedialog.askopenfilenames(title="select files to merge");

#     file2 = filedialog.askopenfilenames(title="select files to merge");

#     output\_file = filedialog.asksaveasfilename(title="save merged file as", defaultextension=".merged")

#     with open(output\_file, 'wb') as f:

#         for file in [file1, file2]:

#             with open(file, 'rb') as infile:

#                 shutil.copyfileobj(infile, f)

#     tk.messagebox.showinfo(title="done", message=f"merged files {file1} and {file2} into {output\_file}")

#     # print(f"merged files {file1} and {file2} into {output\_file}")

Def open\_folder():

   folder = fd.askdirectory(title="select folder to open")

   os.startfile(folder)

Def delete\_folder():

   folder\_to\_delete = fd.askdirectory(title='choose a folder to delete')

   os.rmdir(folder\_to\_delete)

   mb.showinfo("folder deleted", "your desired folder has been deleted")

Def move\_folder():

   folder\_to\_move = fd.askdirectory(title='select the folder you want to move')

   mb.showinfo(message='you just selected the folder to move, now please select the desired destination where you '

                       'want to move the folder to')

   destination = fd.askdirectory(title='where to move the folder to')

   try:

       shutil.move(folder\_to\_move, destination)

       mb.showinfo("folder moved", 'your desired folder has been moved to the location you wanted')

   except:

       mb.showerror('error', 'we could not move your folder. Please make sure that the destination exists')

Def list\_files\_in\_folder():

   i = 0

   folder = fd.askdirectory(title='select the folder whose files you want to list')

   files = os.listdir(os.path.abspath(folder))

   list\_files\_wn = toplevel(root)

   list\_files\_wn.title(f'files in {folder}')

   list\_files\_wn.geometry('450x450')

   list\_files\_wn.resizable(0, 0)

   listbox = listbox(list\_files\_wn, selectbackground='steelblue', font=("georgia", 10))

   listbox.place(relx=0, rely=0, relheight=1, relwidth=1)

   scrollbar = scrollbar(listbox, orient=vertical, command=listbox.yview)

   scrollbar.pack(side=right, fill=y)

   listbox.config(yscrollcommand=scrollbar.set)

   while i < len(files):

       listbox.insert(end, files[i])

       i += 1

Def split\_file():

    file\_path = filedialog.askopenfilename(title="select file to split")

    chunk\_size = tk.simpledialog.askinteger("chunk size", "enter chunk size in mb", minvalue=1)

    chunk\_size \*= 1024 \* 1024

    file\_name, file\_extension = os.path.splitext(file\_path)

    print(os.path.splitext(file\_path))

    chunk\_num = 1

    try:

        with open(file\_path, "rb") as f:

            chunk = f.read(chunk\_size)

            while chunk:

                chunk\_file = f"{file\_name}\_{chunk\_num}{file\_extension}"

                with open(chunk\_file, "wb") as chunk\_f:

                    chunk\_f.write(chunk)

                chunk\_num += 1

                chunk = f.read(chunk\_size)

    except:

        print("please select the file")

    root = tk.tk()

    root.withdraw()

    tk.messagebox.showinfo(title="done", message="file split successfully.")

Def search\_files(root\_path, search\_query, search\_criteria):

    results = []

    for dirpath, dirnames, filenames in os.walk(root\_path):

        for filename in filenames:

            if search\_criteria == 'name':

                if search\_query in filename:

                    results.append(os.path.join(dirpath, filename))

            elif search\_criteria == 'extension':

                if search\_query in filename.split('.')[-1]:

                    results.append(os.path.join(dirpath, filename))

            elif search\_criteria == 'size':

                file\_path = os.path.join(dirpath, filename)

                ans=os.path.getsize(file\_path)\*1024

                if ans <= int(search\_query):

                    results.append(file\_path)

            elif search\_criteria == 'date\_modified':

                file\_path = os.path.join(dirpath, filename)

                modified\_time = os.path.getmtime(file\_path)

                modified\_time = datetime.fromtimestamp(modified\_time)

                if search\_query in modified\_time.strftime('%y-%m-%d'):

                    results.append(file\_path)

    return results

Def search():

    search\_criteria\_var = stringvar()

    search\_query\_var = stringvar()

    search\_frame = frame(root)

    search\_frame.pack(fill=x)

    search\_criteria\_label = label(search\_frame, text='search criteria:')

    search\_criteria\_label.pack(side=left, padx=5, pady=5)

    search\_criteria\_options = ['name', 'extension', 'size', 'date\_modified']

    search\_criteria\_menu = optionmenu(search\_frame, search\_criteria\_var, \*search\_criteria\_options)

    search\_criteria\_menu.pack(side=left, padx=5, pady=5)

    search\_query\_label = label(search\_frame, text='search query:')

    search\_query\_label.pack(side=left, padx=5, pady=5)

    search\_query\_entry = entry(search\_frame, textvariable=search\_query\_var)

    search\_query\_entry.pack(side=left, padx=5, pady=5)

    # create search results window

    def search():

        search\_results\_window = toplevel(root)

        search\_results\_window.title('search results')

        search\_results\_text = text(search\_results\_window)

        search\_results\_text.pack(fill=both, expand=true)

        current\_path = fd.askdirectory(title="select folder to search")

        search\_criteria = search\_criteria\_var.get()

        search\_query = search\_query\_var.get()

        if search\_criteria == '' or search\_query == '':

            messagebox.showerror('error', 'please select a search criteria and enter a search query.')

            return

        search\_results = search\_files(current\_path, search\_query, search\_criteria)

        if len(search\_results) == 0:

            messagebox.showinfo('search results', 'no results found.')

            return

        search\_results\_text.delete('1.0', 'end')

        for result in search\_results:

            search\_results\_text.insert('end', result + '\n')

    search\_button = button(search\_frame, text='search', command=search)

    search\_button.pack(side=left, padx=5, pady=5)

# defining the variables

Title = 'file manager'

Background = '#00ffff'

Button\_font = ("times new roman", 13)

Button\_background ='#808080'

# initializing the window

Root = tk()

Root.title(title)

Root.geometry('550x550')

Root.resizable(0, 0)

Root.config(bg='#ffff45')

# creating and placing the components in the window

Label(root, text=title, font=("comic sans ms", 15), bg='#eeeed5', wraplength=250).place(x=40, y=0)

Button(root, text='open a file', width=20, font=button\_font, bg=button\_background, command=open\_file).place(x=50, y=50)

Button(root, text='copy a file', width=20, font=button\_font, bg=button\_background, command=copy\_file).place(x=50, y=90)

Button(root, text='rename a file', width=20, font=button\_font, bg=button\_background, command=rename\_file).place(x=50, y=130)

Button(root, text='delete a file', width=20, font=button\_font, bg=button\_background, command=delete\_file).place(x=50, y=170)

Button(root, text='open a folder', width=20, font=button\_font, bg=button\_background, command=open\_folder).place(x=50, y=210)

Button(root, text='delete a folder', width=20, font=button\_font, bg=button\_background, command=delete\_folder).place(x=50, y=250)

Button(root, text='move a folder', width=20, font=button\_font, bg=button\_background, command=move\_folder).place(x=50, y=290)

Button(root, text='list all files in a folder', width=20, font=button\_font, bg=button\_background,command=list\_files\_in\_folder).place(x=50, y=330)

Button(root,text='sort all the files',width=20,font=button\_font,bg=button\_background,command=sort\_files).place(x=50,y=370)

Button(root,text='split the file',width=20,font=button\_font,bg=button\_background,command=split\_file).place(x=50,y=410)

Button(root,text='merge files',width=20,font=button\_font,bg=button\_background,command=merge\_files).place(x=50,y=450)

Button(root,text='search\_files',width=20,font=button\_font,bg=button\_background,command=search).place(x=50,y=500)

# finalizing the window

Root.update()

Root.mainloop()