from abc import ABC,abstractmethod

class Bank:

  @abstractmethod

  def check\_balance(ABC):

    pass

  @abstractmethod

  def deposit(ABC):

    pass

  @abstractmethod

  def withdraw(ABC):

    pass

class BankAccount(Bank):

  def \_\_init\_\_(self,balance=0):

    self.balance=balance

  def check\_balance(self):

    return f"Your current balance is:{self.balance}"

  def deposit(self,amount):

    self.balance+=amount

    return "Deposited Successfully"

  def withdraw(self,amount):

    if amount<=self.balance:

      self.balance-=amount

      return "Withdrawn Successfully"

    else:

      return f"Insufficient Funds.Cannot Withdraw"

def main():

  while True:

    print("====================KanwalShehz Bank=========================")

    b=BankAccount(20000)

    option = input("Select\n(1)Check Balance\n(2)Deposit\n(3)Withdraw\n(4)Exit\n")

    if option == '1':

      print(b.check\_balance())

    elif option == '2':

      amount=float(input("Enter amount you want to deposit:"))

      print(b.deposit(amount))

    elif option == '3':

      amount=float(input("Enter amount you want to withdraw:"))

      print(b.withdraw(amount))

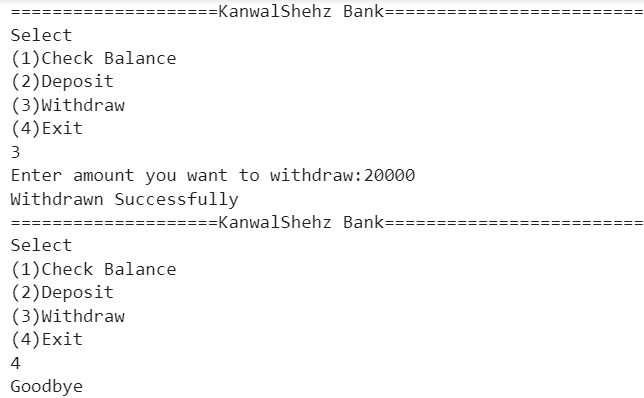
    elif option == '4':

      print("Goodbye")

      break

if \_\_name\_\_ == "\_\_main\_\_":

    main()



import gradio as gr

class LibrarySystem:

    def \_\_init\_\_(self):

        self.catalog = {}

        self.borrowed\_books = {}

    def add\_book(self, title, author, quantity):

        if title in self.catalog:

            self.catalog[title]['quantity'] += quantity

        else:

            self.catalog[title] = {'author': author, 'quantity': quantity}

    def search\_catalog(self, keyword):

        results = []

        for title, details in self.catalog.items():

            if keyword.lower() in title.lower() or keyword.lower() in details['author'].lower():

                results.append({'title': title, 'author': details['author'], 'quantity': details['quantity']})

        return results

    def borrow\_book(self, title, quantity):

        if title in self.catalog and self.catalog[title]['quantity'] >= quantity:

            if title in self.borrowed\_books:

                self.borrowed\_books[title]['quantity'] += quantity

            else:

                self.borrowed\_books[title] = {'author': self.catalog[title]['author'], 'quantity': quantity}

            self.catalog[title]['quantity'] -= quantity

            return f"{title} borrowed successfully"

        else:

            return f"{title} cannot be borrowed"

def library\_interface(action, title='', author='', quantity=0, search\_keyword='', borrow\_title='', borrow\_quantity=0):

    if action == 'Add':

        obj.add\_book(title, author, quantity)

        return f"Book '{title}' added successfully."

    elif action == 'Search':

        search\_results = obj.search\_catalog(search\_keyword)

        return search\_results

    elif action == 'Borrow':

        borrow\_status = obj.borrow\_book(borrow\_title, borrow\_quantity)

        return borrow\_status

obj = LibrarySystem()

iface = gr.Interface(

    fn=library\_interface,

    inputs=[

        gr.Dropdown(['Add', 'Search', 'Borrow'], label='Select Action'),

        gr.Textbox(label='Title'),

        gr.Textbox(label='Author'),

        gr.Number(label='Quantity'),

        gr.Textbox(label='Search Keyword'),

        gr.Textbox(label='Borrow Title'),

        gr.Number(label='Borrow Quantity')

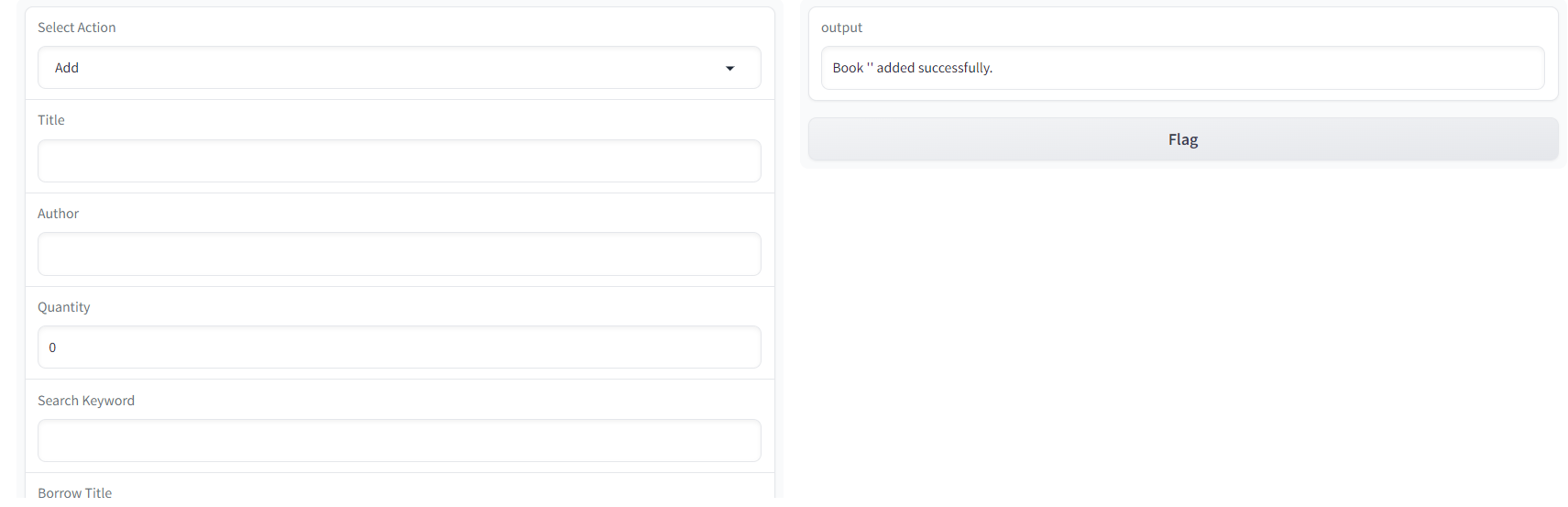
    ],

    outputs="text",

    live=True

)

iface.launch()



import os

import gradio as gr

def recursive\_file\_search(directory, file\_type):

    matching\_files = []

    for root, \_, files in os.walk(directory):

        for file in files:

            if file.endswith(file\_type):

                matching\_files.append(os.path.join(root, file))

    return matching\_files

def file\_search\_interface(directory, file\_type):

    matching\_files = recursive\_file\_search(directory, file\_type)

    if not matching\_files:

        result = "No matching files found."

    else:

        result = "\n".join(matching\_files)

    return result

iface = gr.Interface(

    fn=file\_search\_interface,

    inputs=[

        gr.Textbox(label="Directory", placeholder="Enter the directory path"),

        gr.Textbox(label="File Type", placeholder="Enter the file type (e.g., .txt)")

    ],

    outputs=gr.Textbox(label="Matching Files"),

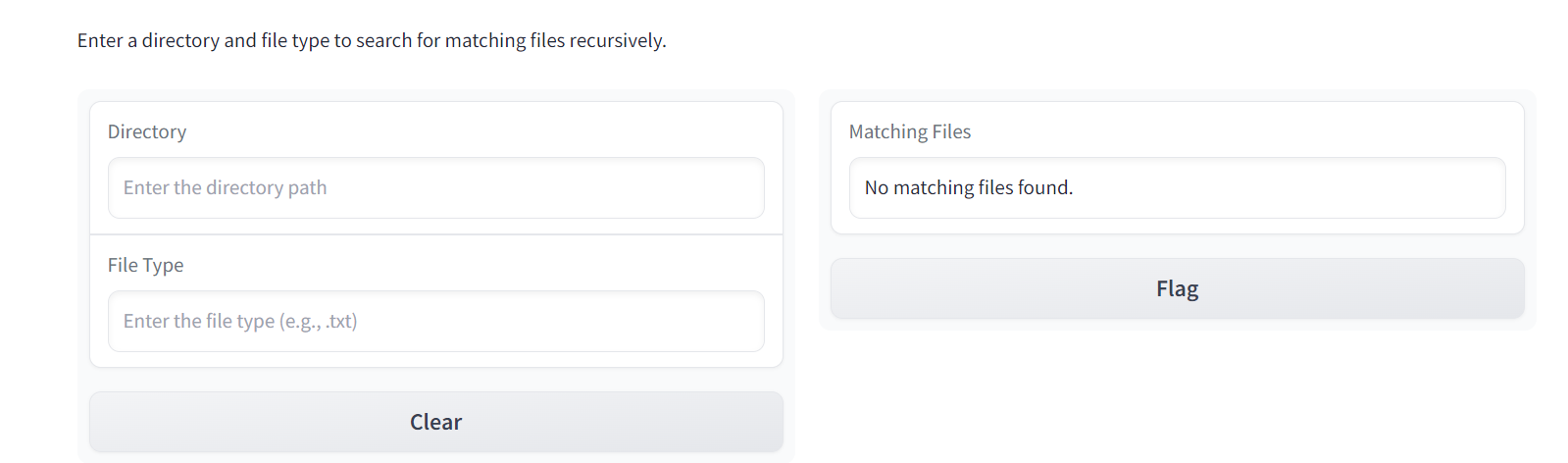
    live=True,

    title="Recursive File Search Interface",

    description="Enter a directory and file type to search for matching files recursively."

)

iface.launch()



import gradio as gr

def calculator(num1, oper, num2):

  num1=float(num1)

  num2=float(num2)

  try:

        if oper == "+":

            return num1 + num2

        elif oper == "-":

            return num1 - num2

        elif oper == "\*":

            return num1 \* num2

        elif oper == "/":

            return round(num1 / num2, 2)

  except Exception as e:

        return f"An unexpected error occurred: {e}"

iface = gr.Interface(fn=calculator,

                     inputs=[

                        gr.Textbox(label="Enter First Number",type="text"),

                        gr.Radio(["+","-","\*","/"],label="Select Operator"),

                        gr.Textbox(label="Enter Second Number",type="text")

                        ],

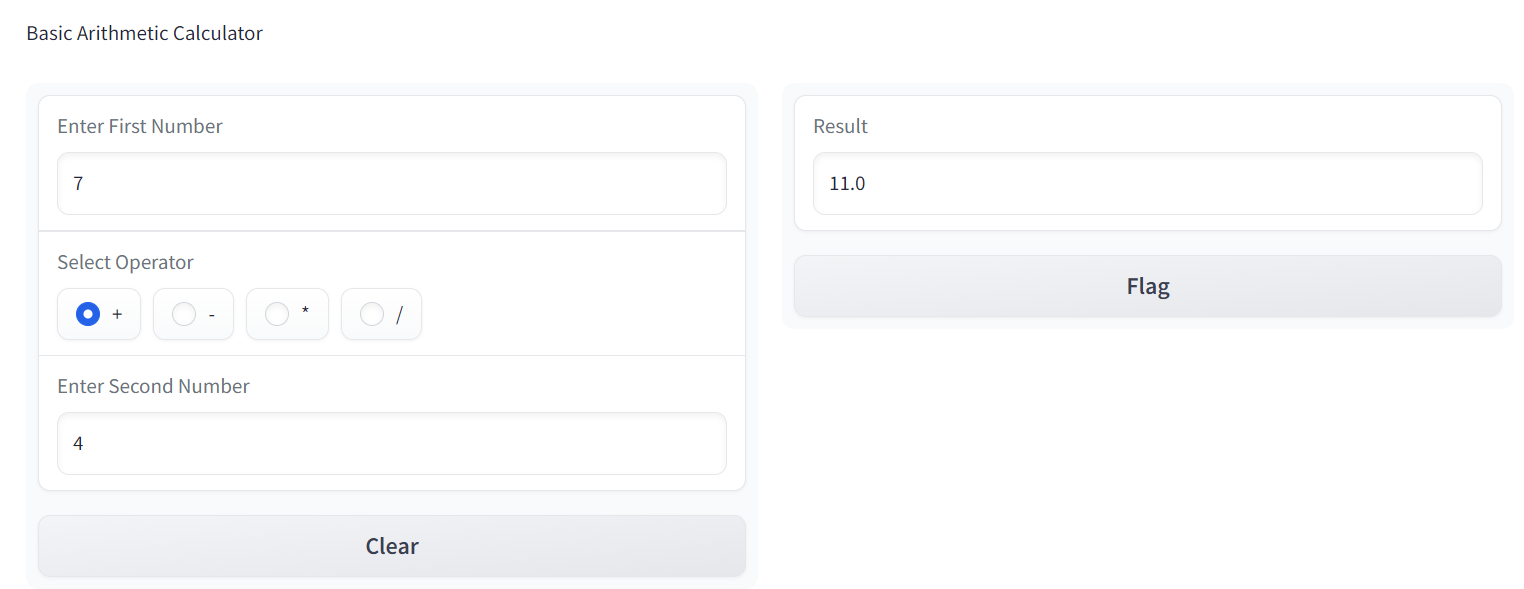
                     outputs=gr.Textbox(label="Result"),

                     live=True,

                     title="Gradio Calculator",

                     description="Basic Arithmetic Calculator")

iface.launch()



class Node:

    def \_\_init\_\_(self, key):

        self.left = None

        self.right = None

        self.val = key

def in\_order\_traversal(root):

    result = []

    if root:

        result += in\_order\_traversal(root.left)

        result.append(root.val)

        result += in\_order\_traversal(root.right)

    return result

def pre\_order\_traversal(root):

    result = []

    if root:

        result.append(root.val)

        result += pre\_order\_traversal(root.left)

        result += pre\_order\_traversal(root.right)

    return result

def post\_order\_traversal(root):

    result = []

    if root:

        result += post\_order\_traversal(root.left)

        result += post\_order\_traversal(root.right)

        result.append(root.val)

    return result

root = Node(1)

root.left = Node(2)

root.right = Node(3)

root.left.left = Node(4)

root.left.right = Node(5)

in\_order\_result = in\_order\_traversal(root)

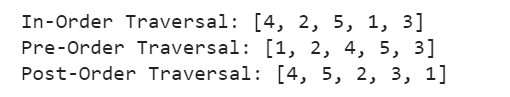
pre\_order\_result = pre\_order\_traversal(root)

post\_order\_result = post\_order\_traversal(root)

print("In-Order Traversal:", in\_order\_result)

print("Pre-Order Traversal:", pre\_order\_result)

print("Post-Order Traversal:", post\_order\_result)



import os

class File:

    def \_\_init\_\_(self, name, size):

        self.name = name

        self.size = size

class Directory:

    def \_\_init\_\_(self, name, contents):

        self.name = name

        self.contents = contents

def calculate\_total\_size(node):

    if isinstance(node, File):

        return node.size

    elif isinstance(node, Directory):

        total\_size = 0

        for content in node.contents:

            total\_size += calculate\_total\_size(content)

        return total\_size

file1 = File("file1.txt", 10)

file2 = File("file2.txt", 5)

file3 = File("file3.txt", 8)

subdirectory1 = Directory("subdirectory1", [file2, file3])

root = Directory("root", [file1, subdirectory1])

total\_size = calculate\_total\_size(root)

print(f"Total size of the directory structure: {total\_size} bytes")

