**Task No. 1:** Banking Interface

Implement a command-line application using interface for a banking system.

Details: Students will create app where users can type commands to check balances, deposit, and withdraw money.

**Solution:**

import gradio as gr

class BankingSystem:

    def \_\_init\_\_(self):

        self.accounts = {}

    def create\_account(self, account\_number, initial\_balance=0):

        self.accounts[account\_number] = initial\_balance

        return f"Account {account\_number} created with initial balance ${initial\_balance}"

    def check\_balance(self, account\_number):

        balance = self.accounts.get(account\_number, "Account not found")

        return f"Balance for account {account\_number}: ${balance}"

    def deposit(self, account\_number, amount):

        if account\_number in self.accounts:

            self.accounts[account\_number] += amount

            return f"${amount} deposited into account {account\_number}\n{self.check\_balance(account\_number)}"

        else:

            return "Account not found"

    def withdraw(self, account\_number, amount):

        if account\_number in self.accounts:

            if self.accounts[account\_number] >= amount:

                self.accounts[account\_number] -= amount

                return f"${amount} withdrawn from account {account\_number}\n{self.check\_balance(account\_number)}"

            else:

                return "Insufficient funds"

        else:

            return "Account not found"

banking\_system = BankingSystem()

def banking\_interface(command, account\_number=None, amount=None, initial\_balance=None):

    if command == 'create':

        return banking\_system.create\_account(account\_number, initial\_balance)

    elif command == 'balance':

        return banking\_system.check\_balance(account\_number)

    elif command == 'deposit':

        return banking\_system.deposit(account\_number, amount)

    elif command == 'withdraw':

        return banking\_system.withdraw(account\_number, amount)

    else:

        return "Invalid command. Please try again."

iface = gr.Interface(

    fn=banking\_interface,

    inputs=[

        gr.Textbox(type="text", label="Command"),

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

        gr.Textbox(type="text", label="Amount/Initial Balance"),

        gr.Textbox(type="text", label="Withdrawal/Deposit Amount"),

    ],

    outputs=gr.Textbox(),

)

iface.launch()

**Output:**



**Task No. 2:** Gradio Library Management System

Objective: Design a Gradio interface for managing a library system. Details: Using Gradio, students will develop a GUI for entering new books, searching the catalog, and managing borrowings.

**Solution:**

import gradio as gr

# Placeholder data structures to simulate a library catalog

library\_catalog = []

borrowed\_books = []

# Function to add a new book to the catalog

def add\_book(title, author, ISBN):

    book = {'title': title, 'author': author, 'ISBN': ISBN}

    library\_catalog.append(book)

    return f"Book '{title}' by {author} added to the catalog."

# Function to search the catalog based on the query

def search\_catalog(query):

    results = [book['title'] for book in library\_catalog if query.lower() in book['title'].lower()]

    return results if results else "No matching books found."

# Gradio Interface for adding a book

add\_book\_interface = gr.Interface(

    fn=add\_book,

    inputs=["text", "text", "text"],

    outputs="text",

    live=True,

    title="Add a New Book to the Catalog"

)

# Gradio Interface for searching the catalog

search\_catalog\_interface = gr.Interface(

    fn=search\_catalog,

    inputs="text",

    outputs="text",

    live=True,

    title="Search the Library Catalog"

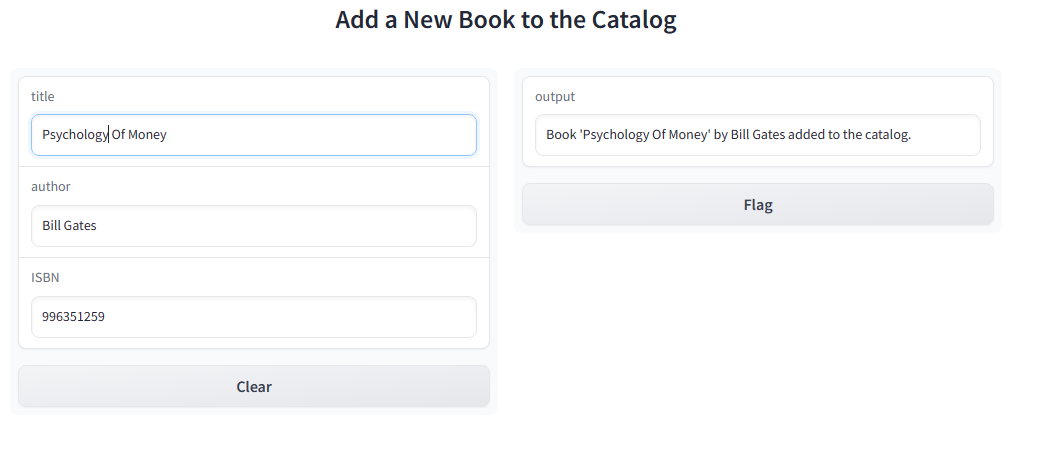
)

# Launch Gradio Interfaces

add\_book\_interface.launch()

search\_catalog\_interface.launch()

**Output:**



A screenshot of a computer

Description automatically generated

**Task No. 3:** Gradio Recursive File Search Interface

Objective: Create a Gradio interface that allows users to search files recursively.

Details: Students will develop a Gradio app that provides a user interface for specifying a directory and file type, then displays a list of all matching files found through recursive search.

**Solution:**

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 "\n".join(matching\_files)

file\_types = [".txt", ".pdf", ".jpg"] # Add more file types as needed

iface = gr.Interface(

fn=recursive\_file\_search,

inputs=[gr.Textbox(), gr.Dropdown(file\_types)],

outputs=gr.Textbox(),

live=True,

title="Recursive File Search",

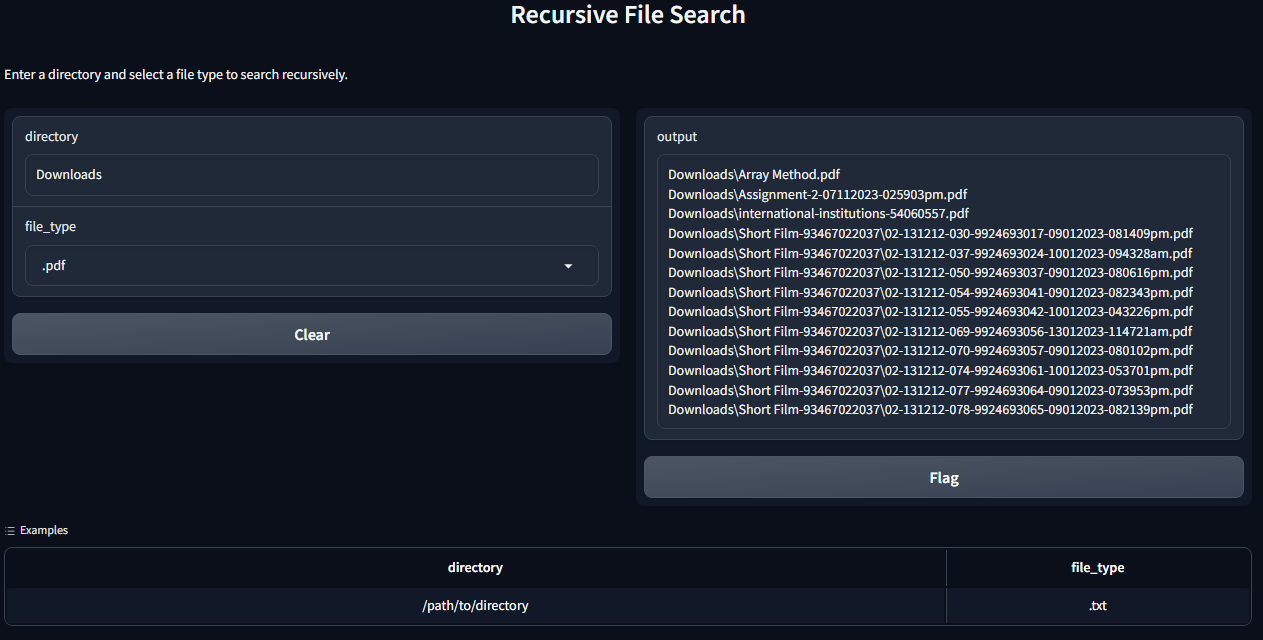
description="Enter a directory and select a file type to search recursively.",

examples=[["/path/to/directory", ".txt"]]

)

iface.launch()

**Output:**

****

**Task No. 4:** Gradio-based Calculator

Objective: Build a calculator with a graphical user interface using Gradio.

Details: Students will use Gradio to create a web-based calculator that performs basic arithmetic operations and displays the results on the GUI.

**Solution:**

import gradio as gr

# Define the arithmetic operations

def add(a, b):

    return a + b

def subtract(a, b):

    return a - b

def multiply(a, b):

    return a \* b

def divide(a, b):

    if b != 0:

        return a / b

    else:

        return "Cannot divide by zero"

# Define the Gradio interface

iface = gr.Interface(

    fn=add,  # You can choose any operation here: add, subtract, multiply, divide

    inputs=[

        gr.Number(label="Enter first number"),

        gr.Number(label="Enter second number"),

    ],

    outputs=gr.Textbox(),

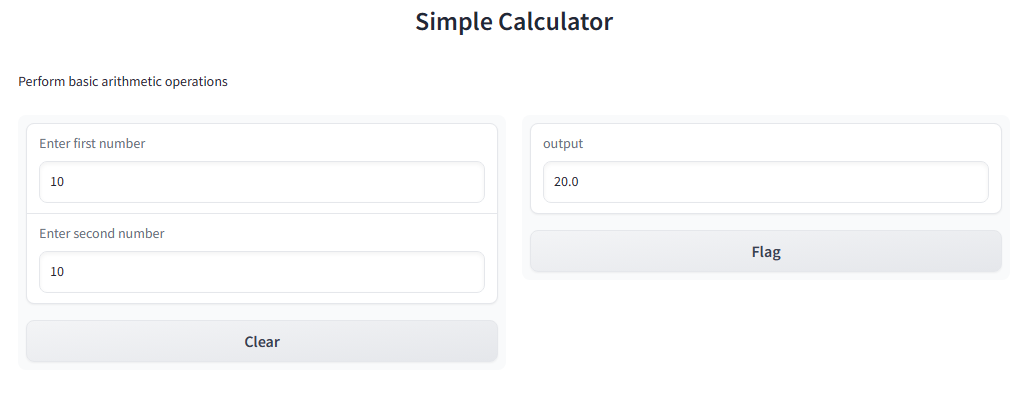
    live=True,

    title="Simple Calculator",

    description="Perform basic arithmetic operations",

)

iface.launch()

**Output:**

**Task No. 5:** Binary Tree Traversal

Create a Python program that defines a binary tree as a recursive data type. Implement functions for in-order, pre-order, and post-order traversals of the binary tree. Test your program with a sample binary tree and display the results of each traversal.

**Solution:**

import gradio as gr

class TreeNode:

    def \_\_init\_\_(self, value):

        self.value = value

        self.left = None

        self.right = None

def in\_order\_traversal(node, result):

    if node is not None:

        in\_order\_traversal(node.left, result)

        result.append(node.value)

        in\_order\_traversal(node.right, result)

def pre\_order\_traversal(node, result):

    if node is not None:

        result.append(node.value)

        pre\_order\_traversal(node.left, result)

        pre\_order\_traversal(node.right, result)

def post\_order\_traversal(node, result):

    if node is not None:

        post\_order\_traversal(node.left, result)

        post\_order\_traversal(node.right, result)

        result.append(node.value)

def traverse\_binary\_tree(traversal\_type):

    # Sample binary tree

    root = TreeNode(1)

    root.left = TreeNode(2)

    root.right = TreeNode(3)

    root.left.left = TreeNode(4)

    root.left.right = TreeNode(5)

    root.right.left = TreeNode(6)

    root.right.right = TreeNode(7)

    result = []

    if traversal\_type == "In-order":

        in\_order\_traversal(root, result)

    elif traversal\_type == "Pre-order":

        pre\_order\_traversal(root, result)

    elif traversal\_type == "Post-order":

        post\_order\_traversal(root, result)

    return result

# Gradio Interface

iface = gr.Interface(

    fn=traverse\_binary\_tree,

    inputs="text",

    outputs="text",

    live=True,

    examples=[["In-order"], ["Pre-order"], ["Post-order"]]

)

iface.launch()

**Output:**

**A screenshot of a computer

Description automatically generated**

**Task No. 6:** Directory Structure Analysis

Design a program that models a directory structure using a recursive data type. Each node in the structure represents either a file or a directory. Implement a function that calculates the total size of the directory, considering the sizes of all files and subdirectories. Test your program with a sample directory structure and display the total size.

**Solution:**

import gradio as gr

import os

def calculate\_total\_size(directory\_path):

try:

total\_size = sum(os.path.getsize(os.path.join(dirpath, filename))

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

for filename in filenames)

return f"Total size of the directory '{directory\_path}': {total\_size} bytes"

except FileNotFoundError:

return f"Directory '{directory\_path}' not found."

except PermissionError:

return f"Permission denied for directory '{directory\_path}'."

except Exception as e:

return f"An error occurred: {e}"

iface = gr.Interface(

fn=calculate\_total\_size,

inputs=gr.Textbox("text", label="Enter directory path:"),

outputs="text",

live=True

)

iface.launch()

**Output:**

