Bahria University,

Karachi Campus



LAB EXPERIMENT NO.

\_\_\_08\_\_\_

LIST OF TASKS

|  |  |
| --- | --- |
| TASK NO | OBJECTIVE |
| 01 | 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. |
| 02 | **Details**: Using Gradio, students will develop a GUI for entering new books, searching the catalog, and managing borrowings. |
| 03 | 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.. |
| 04 | 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. |
| 05 | 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. |
| 06 | 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. |

Submitted On

29-11-2023

(Date: DD/MM/YY)

**Task 01:** 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 random

class BankingSystem:

def \_\_init\_\_(self):

self.card\_numbers = []

self.pin\_numbers = []

def main\_welcome\_screen(self):

print("\n1. Create an account\n2. Log into account\n0. Exit")

main\_menu\_selection = input("Select Your Option (1-2, or 0 to exit): ")

if main\_menu\_selection == "1":

self.create\_account()

elif main\_menu\_selection == "2":

self.account\_login()

elif main\_menu\_selection == "0":

print("\nBye!")

return

def create\_account(self):

credit\_card\_number = "400000" + format(random.randint(0000000000, 9999999999), '010d')

pin\_number = format(random.randint(0000, 9999), '04d')

self.card\_numbers.append(credit\_card\_number)

self.pin\_numbers.append(pin\_number)

print("\nYour card has been created")

print("Your card number:", credit\_card\_number)

print("Your pin number:", pin\_number)

self.main\_welcome\_screen()

def account\_login(self):

entered\_card\_number = input("\nEnter your card number: ")

entered\_pin\_number = input("Enter your PIN: ")

if entered\_card\_number in self.card\_numbers and entered\_pin\_number in self.pin\_numbers:

print("\nYou have successfully logged in!")

self.account\_balance()

else:

print("\nWrong card number or PIN!")

self.main\_welcome\_screen()

def account\_balance(self):

print("\n1. Balance\n2. Log out\n0. Exit")

account\_balance\_selection = input("Select Your Option (1-2, or 0 to exit): ")

banking\_system = BankingSystem()

banking\_system.main\_welcome\_screen()

A screenshot of a computer program

Description automatically generated**Output:**

**Task 02:** **Details**: Using Gradio, students will develop a GUI for entering new books, searching the catalog, and managing borrowings.

**Solution:**

import gradio as gr

def add\_book(title, author, publication\_date):

return "Book added successfully"

def search\_catalog(query):

return "Search results based on the query"

def manage\_borrowings(book\_id, student\_id, action):

return "Borrowing action performed successfully"

add\_book\_interface = gr.Interface(fn=add\_book, inputs=["text", "text", "text"], outputs="text", title="Add New Book")

search\_catalog\_interface = gr.Interface(fn=search\_catalog, inputs="text", outputs="text", title="Search Catalog")

manage\_borrowings\_interface = gr.Interface(fn=manage\_borrowings, inputs=["text", "text", "text"], outputs="text", title="Manage Borrowings")

add\_book\_interface.launch()

search\_catalog\_interface.launch()

manage\_borrowings\_interface.launch()

**Output:**

**A screenshot of a search catalog

Description automatically generated**A screenshot of a computer

Description automatically generated

**A screenshot of a computer

Description automatically generated**

**Task 03:** 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 gradio as gr

import os

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

matching\_files = []

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

for file in files:

if file.endswith(file\_extension):

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

return matching\_files

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

directory = directory if directory else "sample\_directory"

file\_extension = file\_extension if file\_extension else ".txt"

sample\_directory = os.path.join(os.getcwd(), directory)

sample\_file\_extension = file\_extension

matching\_files = recursive\_file\_search(sample\_directory, sample\_file\_extension)

if not matching\_files:

result = "No matching files found in the sample directory."

else:

result = "Matching files in the sample directory:\n" + "\n".join(matching\_files)

return result

iface = gr.Interface(

fn=file\_search\_interface,

inputs=[

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

gr.Textbox(type="text", label="File Extension"),

],

outputs=gr.Textbox(),

live=True,

title="Recursive File Search",

description="Enter the directory and file extension to search for matching files recursively.",

)

face.launch()

**Output:**

A screenshot of a computer

Description automatically generated

**Task 04:** 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

def calculator(num1, operation, num2):

if operation == "add":

return num1 + num2

elif operation == "subtract":

return num1 - num2

elif operation == "multiply":

return num1 \* num2

elif operation == "divide":

if num2 == 0:

raise gr.Error("Cannot divide by zero!")

return num1 / num2

iface = gr.Interface(

fn=calculator,

inputs=["number", gr.Radio(["add", "subtract", "multiply", "divide"]), "number"],

outputs="number",

title="Simple Calculator",

description="Perform basic arithmetic operations"

)

iface.launch()

**Output:**

A screenshot of a computer

Description automatically generated

**Task 05:** 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:**

class Node:

def \_\_init\_\_(self, data):

self.left = None

self.right = None

self.data = data

def inorder\_traversal(node):

if node:

inorder\_traversal(node.left)

print(node.data, end=" ")

inorder\_traversal(node.right)

def preorder\_traversal(node):

if node:

print(node.data, end=" ")

preorder\_traversal(node.left)

preorder\_traversal(node.right)

def postorder\_traversal(node):

if node:

postorder\_traversal(node.left)

postorder\_traversal(node.right)

print(node.data, end=" ")

root = Node(1)

root.left = Node(2)

root.right = Node(3)

root.left.left = Node(4)

root.left.right = Node(5)

print("Inorder traversal:")

inorder\_traversal(root)

print("\nPreorder traversal:")

preorder\_traversal(root)

print("\nPostorder traversal:")

postorder\_traversal(root)

**Output:**

A screenshot of a computer

Description automatically generated

**Task 06:** 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:**

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\_directory\_size(node):

if isinstance(node, File):

return node.size

elif isinstance(node, Directory):

total\_size = 0

for content in node.contents:

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

return total\_size

file1 = File("file1.txt", 100)

file2 = File("file2.txt", 150)

file3 = File("file3.txt", 200)

subdir1 = Directory("subdir1", [file1, file2])

subdir2 = Directory("subdir2", [file3])

main\_directory = Directory("main\_directory", [subdir1, subdir2])

total\_size = calculate\_directory\_size(main\_directory)

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

**Output:**

