TRIBHUVAN UNIVERSITY

Institute of Engineering

Pulchowk Campus

A LAB REPORT ON

Functional Programming Exerceise and Creating Task Manager as Mini Project

LAB NO. : 3

EXPERIMENT DATE : 2025-07-28

SUBMITTED DATE : 2025-08-04

SUBMITTED BY

Name : Abhishek Kharel

Group : A

Roll No. : 081BEL005

SUBMITTED TO

Department of

Computer Engineering

lab3

August 4 , 2025

1 Lab 3

1.1 Functional Programming

1. Write a Python function named greet_user that takes a user's name and prints:

[56]:
```
def greet_user(name):
    print(f"Hello, {name}! Welcome!")

greet_user("Abhishek")
```

```
Hello, Abhishek! Welcome!
```

2. Hello, ! Welcome to Python. Call the function with a sample name.

[57]:
```
def greet_user(name):
    print(f"Hello, {name}! Welcome to Python.")

greet_user("Abhishek")
```

```
Hello, Abhishek! Welcome to Python.
```

3. Create a function power(base, exponent=2) that returns the result of base raised to the power of exponent.Demonstrate it with and without the exponent argument.

[58]:
```
def power(base, exponent=2):
    return base ** exponent

result1 = power(5)
print(f"5 squared = {result1}")

result2 = power(2, 3)
print(f"2 raised to 3 = {result2}")
```

```
5 squared = 25
2 raised to 3 = 8
```

4. Write a function book_info(title, author, year) that prints book details.Call the function using keyword arguments in different orders.

[59]:
```python
def book_info(title, author, year):
    print(f"Title: {title}")
    print(f"Author: {author}")
    print(f"Year: {year}")

book_info(title="Python Basics", author="John Doe", year=2022)

print()

book_info(year=2021, title="Advanced Python", author="Jane Smith")
```

```
Title: Python Basics
Author: John Doe
Year: 2022

Title: Advanced Python
Author: Jane Smith
Year: 2021
```

5. Create a function sum_numbers(*args) that accepts any number of numeric arguments and returns their sum.Test it with 2, 3, and 5 numbers.

[60]:
```python
def sum_numbers(*args):
    return sum(args)

print("Sum of 2 numbers:", sum_numbers(10, 20))
print("Sum of 3 numbers:", sum_numbers(5, 10, 15))
print("Sum of 5 numbers:", sum_numbers(1, 2, 3, 4, 5))
```

```
Sum of 2 numbers: 30
Sum of 3 numbers: 30
Sum of 5 numbers: 15
```

6. Write a function student_profile(**kwargs) that prints the key-value pairs passed (e.g., name, age, grade). Call it with at least three named arguments.

[61]:
```python
def student_profile(**kwargs):
    print("Student Profile:")
    for key, value in kwargs.items():
        print(f"{key}: {value}")

student_profile(name="Abhishek", age=20, grade="A+")
```

```
Student Profile:
name: Abhishek
age: 20
grade: A+
```

7. Write a lambda function to compute the square of a number.Use it to compute the square of 5 and 12.

[62]:
```
square = lambda x: x ** 2

print("Square of 5:", square(5))
print("Square of 12:", square(12))
```

```
Square of 5: 25
Square of 12: 144
```

8. Given a list of numbers [1, 2, 3, 4, 5], use map() and a lambda function to return a new list with each number doubled.

[63]:
```
numbers = [1, 2, 3, 4, 5]
doubled_numbers = list(map(lambda x: x * 2, numbers))
print(doubled_numbers)
```

```
[2, 4, 6, 8, 10]
```

9. Given a list [10, 15, 20, 25, 30], use filter() and a lambda function to extract numbers divisible by 10.

[64]:
```
numbers = [10, 15, 20, 25, 30]
divisible_by_10 = list(filter(lambda x: x % 10 == 0, numbers))
print(divisible_by_10)
```

```
[10, 20, 30]
```

Given a list of temperatures in Celsius [36.5, 37.0, 39.2, 35.6, 38.7],convert them to Fahrenheit using map(),Filter out those above 100°F using filter().

[65]:
```
celsius = [36.5, 37.0, 39.2, 35.6, 38.7]
fahrenheit = list(map(lambda c: (c * 9/5) + 32, celsius))
high_temps = list(filter(lambda f: f > 100, fahrenheit))
print("Temperatures in Fahrenheit:", fahrenheit)
print("Temperatures above 100°F:", high_temps)
```

```
Temperatures in Fahrenheit: [97.7, 98.6, 102.56, 96.08000000000001, 101.66]
Temperatures above 100°F: [102.56, 101.66]
```

1.2 Mini Project

Simple To-Do Manager Using Functional Programming Objective: Manage a list of to-do tasks using functions, lambda, filter, and map. Requirements: - Allow adding tasks using a function add_task(task_list, task_name). - Each task is a dictionary: { “name”: str, “completed”: bool }. - Use lambda and filter() to list only incomplete tasks. - Use map() to mark all tasks as completed. - Include a search_tasks(task_list, keyword) function using filter() and lambda.

```
Sample Workflow:
tasks = []
tasks = add_task(tasks, "Buy groceries")
tasks = add_task(tasks, "Finish assignment")
tasks = add_task(tasks, "Call friend")
```

```
# List incomplete tasks
print("Pending Tasks:", list_pending(tasks))
```

[66]:
```
def add_task(task_list, task_name):
    task = {"name": task_name, "completed": False}
    return task_list + [task]

def list_pending(task_list):
    return list(filter(lambda t: not t["completed"], task_list))

def complete_all(task_list):
    return list(map(lambda t: {**t, "completed": True}, task_list))

def search_tasks(task_list, keyword):
    return list(filter(lambda t: keyword.lower() in t["name"].lower(),␣
 ↪task_list))

tasks = []
tasks = add_task(tasks, "Buy groceries")
tasks = add_task(tasks, "Finish assignment")
tasks = add_task(tasks, "Call friend")

print("Pending Tasks:", list_pending(tasks))

tasks = complete_all(tasks)
print("All tasks after marking complete:", tasks)

print("Search Result for 'call':", search_tasks(tasks, "call"))
```

```
Pending Tasks: [{'name': 'Buy groceries', 'completed': False}, {'name': 'Finish
assignment', 'completed': False}, {'name': 'Call friend', 'completed': False}]
All tasks after marking complete: [{'name': 'Buy groceries', 'completed': True},
{'name': 'Finish assignment', 'completed': True}, {'name': 'Call friend',
'completed': True}]
Search Result for 'call': [{'name': 'Call friend', 'completed': True}]
```