TRIBHUVAN UNIVERSITY  
Institute of Engineering  
Pulchowk Campus

LAB REPORT 3:

Submitted By:  
Sashank Bhattarai  
081BEL075

Submitted To:  
Department of Computer Engineering

1. **Write a Python function named greet\_user that takes a user's name and prints <Hello, <name>! Welcome to Python.>:**

**Program:**

name=input("Enter your name:")

def greet\_user(name):

    print(f"Hello, {name}! Welcome to Python.")

greet\_user(name)

**Output:**



1. **Call the function with a sample name.**
2. **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.**

**Program:**

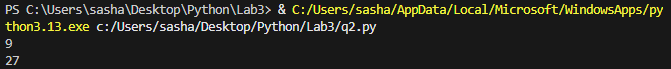
def power(base, exponent=2):

    return base \*\* exponent

print(power(3))

print(power(3, 3))

**Output:**

****

1. **Write a function book\_info(title, author, year) that prints book details. Call the function using keyword arguments in different orders.**

**Program:**

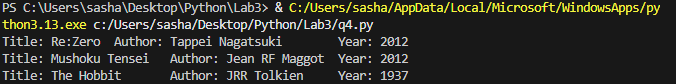
def book\_info(title, author, year):

    print(f"Title: {title}\tAuthor: {author}\tYear: {year}")

book\_info(title="Re:Zero", author="Tappei Nagatsuki", year=2012)

book\_info(author="Jean RF Maggot", year=2012, title="Mushoku Tensei")

book\_info(year=1937, title="The Hobbit", author="JRR Tolkien")

**Output:**

1. **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.**

**Program:**

def sum\_numbers(\*args):

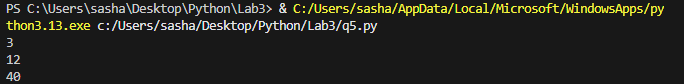
return sum(args)

print(sum\_numbers(1, 2))

print(sum\_numbers(3, 4, 5))

print(sum\_numbers(6, 7, 8, 9, 10))

**Output:**

****

1. **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.**

**Program:**

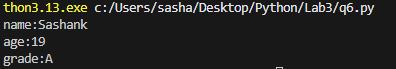
def student\_profile(\*\*kwargs):

    for key, value in kwargs.items():

        print(f"{key}:{value}")

student\_profile(name="Sashank", age=19, grade="A")

**Output:**

****

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

**Program:**

square = lambda x: x \*\* 2

print(square(5))

print(square(12))

**Output:**

****

1. **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.**

**Program:**

numbers = [1, 2, 3, 4, 5]

doubled = list(map(lambda x: x \* 2, numbers))

print(doubled)

**Output:**

****

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

**Program:**

numbers = [10, 15, 20, 25, 30]

div = list(filter(lambda x: x % 10==0, numbers))

print(div)

**Output:**

****

1. **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().**

**Project:**

Celsius = [36.5, 37.0, 39.2, 35.6, 38.7]

Fahrenheit = list(map(lambda c: c \* 9/5 + 32, Celsius))

below100F = list(filter(lambda f: f <= 100, Fahrenheit))

print("Fahrenheit temperatures:", Fahrenheit)

print("Temperatures at or below 100°F:", below100F)

**Output:**

****

**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))

# Mark all tasks as complete

tasks = complete\_all(tasks)

# Search tasks with keyword "call"

print("Search Result:", search\_tasks(tasks, "call"))

**PROGRAM:**

# Add a task

def add\_task(task\_list, task\_name):

    task = {"name": task\_name, "completed": False}

    task\_list.append(task)

    return task\_list

# List pending

def list\_pending(task\_list):

    return list(filter(lambda t: not t["completed"], task\_list))

# Search tasks by keyword

def search\_tasks(task\_list, keyword):

    return list(filter(lambda t: keyword.lower() in t["name"].lower(), task\_list))

# Mark a single task as completed by index

def complete\_task(task\_list, index):

    if 0 <= index < len(task\_list):

        task\_list[index]["completed"] = True

    else:

        print("Invalid task number!")

    return task\_list

# Display task

def show\_tasks(task\_list):

    if not task\_list:

        print("No tasks found.")

        return

    for i, t in enumerate(task\_list, 1):

        status = "Done" if t["completed"] else "Pending"

        print(f"{i}. {t['name']} - {status}")

#

# Main Workflow

#

tasks = []

tasks = add\_task(tasks, "Walk")

tasks = add\_task(tasks, "Homework")

tasks = add\_task(tasks, "Run")

n = int(input("How many tasks do you want to add? "))

for i in range(n):

    name = input(f"Enter task {i+1}: ")

    tasks = add\_task(tasks, name)

# Mark a single task as completed

print("\nAll Tasks:")

show\_tasks(tasks)

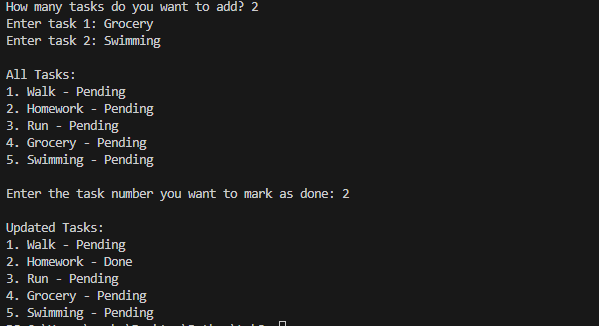
choice = int(input("\nEnter the task number you want to mark as done: ")) - 1

tasks = complete\_task(tasks, choice)

print("\nUpdated Tasks:")

show\_tasks(tasks)

**OUTPUT:**

****