Course Code	Course Title	L	T	P	C
BACSE101	Problem Solving Using Python	0	0	4	2
Pre-requisite	NIL	Syllabus Version			
		1			

Course Objectives

1. To impart algorithmic skills through python programming language fundamentals and constructs.
2. To explore python collections for efficient data handling techniques and to learn modular, recursive programming approaches for developing software applications.
3. To understand python libraries for data analysis and manipulations

Course Outcomes

1. Identify appropriate algorithmic approach, data representation, control constructs in developing software solutions for solving real-world problems.
2. Inculcate data handling techniques using python collections and modular programming strategies for developing multi-disciplinary software applications
3. Idealize the importance of python libraries for handling, manipulating and analyzing multi-faceted real world problem domains.

Indicative Experiments

1. **Subsidy Computation** Agricultural subsidies are offered by the Indian government to farmers for improving crop yields and benefiting them. To facilitate the farmers in finding the subsidy amount, develop a Python code to calculate the subsidy based on the crop the farmer grows. The farmers can grow one of these three types of crops: Rice, Wheat, or Maize. The program calculates and displays the type of crop, total subsidy based on the type of crop selected, along with the yield and market rate. Use menu driven approach to implement the same. Each crop has a different formula for calculating the subsidy: Rice: Subsidy= (Yield × Market Rate) +100 Wheat: Subsidy= (Yield × Market Rate) +200 Maize: Subsidy= (Yield × Market Rate) +300 Input Format Press 1/2/3 to compute the Subsidy of Rice/Wheat/Maize Next Line to enter the Market Rate Next Line to enter the Yield Enter either Yes/No .	4 hours

Press 1/2/3 to compute the Subsidy of Rice/Wheat/Maize Next Line to enter the Market Rate Next Line to enter the Yield Enter either Yes/No Output Format Display “Press 1/2/3 to compute the Subsidy of Rice/Wheat/Maize” Display the Type of Crop, Subsidy, Yield and Market Rate Do you want to continue (Yes/No). If Yes . Press 1/2/3 to compute the Subsidy of Rice/Wheat/Maize Display the Type of Crop, Subsidy, Yield and MarketRate Do you want to continue (Yes/No). If No, display ‘Thank you’ **Climate Change Data Analyzer** Scientists track daily temperatures as a string of values separated by spaces (e.g., "30.5 32.1 29.8"). Write a python program that extracts the temperatures, converts them into float values, and finds the highest, lowest, and average temperature. Rules & Constraints: • Input is a string of space-separated float values (e.g., "30.5 31.2"). • All values must be valid floating-point numbers. • The list must have at least two values. • Temperatures must be within a realistic range: -100.0°C to 60.0°C. • The program should round the average to two decimal places. • If the input is invalid (non-numeric, empty, out-of-range), return an error message. Input Format: A single string with space-separated temperature readings (e.g., "25.4 27.8 26.5") Output Format: Highest: Lowest: Average:	
2. **Gaming Leaderboard Username Validator** In an online game, players choose usernames for the leaderboard. To make sure all usernames are clean and fair, they must follow these rules: • The username must start with a letter (A–Z or a–z). • It can contain letters, numbers, and underscores (_) only. • It cannot start with a number.	4 hours

• No other symbols or spaces are allowed. • The length must be between 5 and 15 characters. Write a python program using regular expressions to check if a username is valid or invalid. If invalid, display a message “Invalid Name” Input Format: A single string – the username to validate. Output Format: Print "valid username" if it follows all rules. Otherwise, print "Invalid Name". **Number Game** In Numeria, Alchemists must build an arithmetic expression using given numbers and operations to match a target. Use each number once, with +, -, *, or /. If no exact match, return the expression closest to the target. If multiple expressions give the same best result, return "No single solution" Rules: • Use all numbers once. • Only +, -, *, / allowed. • Division must result in integers only. • Left-to-right evaluation only (no brackets) • All numbers must be positive integers. • Return one valid expression, or "No single solution" if ties exist. Input Format: ((num1, num2, ..., numN), target) Output Format: • A string like "2+3*5", or • "No single solution" • Error message for invalid input	
3. **Drone Delivery** A delivery drone needs to find the shortest path from a warehouse (starting point) to a delivery location (destination) in a grid-based city map. The drone can move up, down, left, or right, but it cannot pass through restricted zones (obstacles). Implement an algorithm (only) using a top-down approach to find the shortest path from the warehouse to the delivery location. Input Format: A 2D grid (N × N) where 0 represents open paths and 1 represents obstacles. The starting position (warehouse) and destination position (delivery location). Top- down Design Approach:	6 hours

If the destination is reached, return the path length. If the current cell is an obstacle or out of bounds, return an invalid path value. Try moving in all four directions (up, down, left, right). Use memoization to store results of already visited paths. Output Format: The minimum number of steps required to reach the destination. The shortest path length or an indication that the delivery is not possible. **Task Completion Time** A group of students is participating in a coding competition, where they need to solve a set of problems as quickly as possible. Each student has a recorded completion time (in minutes) for each problem. The goal is to identify the student who solved their problems in the shortest total time. Implement an algorithm using Divide and Conquer problem solving approach to determine the student with the lowest total completion time. **Input:** A list of students with their total completion times. **Base Case:** If there is only one student, return them as the fastest. **Divide**: Split the student list into two halves. **Conquer**: Recursively find the fastest student in both halves. **Combine**: Compare the two selected students and return the one with the lower total time.	
4. **Rental Computation** Maya is developing a pricing system for a photography studio that rents out different types of cameras and needs to calculate the rental cost based on the equipment type and duration. The system should apply discounts based on the rental cost thresholds: Video Camera: The rental rate is ₹250 per hour. If the total rental cost exceeds ₹1500, a 12% discount is applied. Drone Camera: The rental rate is ₹300 per hour. If the total rental cost exceeds ₹2000, a 15% discount is applied. Write a Python code to compute the rentalCost for VideoCamera, and DroneCamera and apply the respective discount based on the rental duration and the type of camera. Input Format First line contains the duration of usage of Video Camera Second line contains the rental rate of Video camera per hour Third line contains the duration of usage of Drone Camera Fourth line contains the rental rate of Drone camera per hour. Output Format First line prints the total rental cost of Video Camera Second line prints the discount for Video Camera usage.	4 hours

Third line prints the total rental cost with discount for Video Camera. Fourth line prints the total rental cost of Drone Camera Fifth line prints the discount for Drone Camera usage. Sixth line Prints the total rental cost with discount for Drone Camera. **Leader Selection** In the city of Solaria, a group of n scholars sit in a circle. Each scholar has a number from 1 to n. The selection for the next Guardian of Harmony works like this: • Scholar 1 tells Scholar 2 to leave the circle. • Then the next scholar (still in the circle) tells the person next to them to leave. • This continues in a circle — each person removes the one next to them — until only one scholar is left. Determine which scholar will be the last one standing Input Format: A single number n — the number of scholars. Output Format: A number — the scholar who is left at the end.	
5. **Analyze health risk** Analyze health risks based on multiple individuals' BMI (Body Mass Index), blood pressure, and heart rate. Using a loop to collect data for multiple individuals. Calculate BMI using BMI = weight / (height ** 2), use conditional statements to classify BMI as low, medium or high. Low BMI (underweight): Less than 18.5 kg/m² Medium BMI (normal weight): 18.5 - 24.9 kg/m² High BMI (overweight): 25 kg/m² and above A "low" blood pressure is generally considered to be below 90/60 mmHg, while a "medium" or normal blood pressure is between 90/60 mmHg and 120/80 mmHg, otherwise high. For heart rate, a "low" resting rate is under 60 beats per minute (bpm), and a "medium" or normal resting rate is between 60-100 bpm, otherwise high. Additionally, check if blood pressure (systolic/diastolic) and heart rate fall within healthy ranges, categorizing individuals into Low, Moderate, or High health risk levels. Write a python code to display the BMI, blood pressure and heart rate. If two or more health risks are high, then the risk level is high. If any one of the health risk is high, then the risk level is medium. Otherwise, it is under low risk. Finally, display a summary report showing the number of individuals in each risk category. Input Format: First line contains the number of People ‘n’	6 hours

Next line contains the Blood pressure(Systolic/Diastolic) of Person1 in mm/hg
Next line contain heartrate of Person1 in beats per minute(bpm)
.
Next line contains the weight of the Person ‘n’ in Kg
Next line contains the height of the Person ‘n’ in meter
Next line contains the Blood pressure(Systolic/Diastolic) of Person ‘n’ in mm/hg
Next line contain heartrate of Person ‘n’ in beats per minute(bpm)

Output Format
First line Prints the BMI of Person 1(kg/m2)
Second line prints underweight/normalweight/overweight of Person 1
Third line Prints the Blood Pressure of Person 1(mm/hg)
Fourth Line prints Low/Medium/HighBloodPressure of Person 1
Fifth line prints the Heart rate of Person 1(bpm)
Sixth line prints low/normal/high heartrate of Person 1
Seventh line prints low/medium/high risk level of Person 1
.
Next line Prints the BMI of Person ‘n’(kg/m2)
Next line prints underweight/normalweight/overweight of Person ‘n’
Next line Prints the Blood Pressure of Person ‘n’(mm/hg)
Next Line prints Low/Medium/HighBloodPressure of Person ‘n’

Next line prints the Heart rate of Person ‘n’(bpm)
Next line prints low/normal/high heartrate of Person ‘n’
Next line prints low/medium/high risk level of Person ‘n’

Next prints the Summary:
High risk level : number of People
Medium risk level: number of People
Low risk level: number of People

Process DNA sequence

DNA sequences are primarily analyzed by scientists, researchers, and medical professionals in fields like genetics, molecular biology, and medicine, to understand genetic variations, diagnose diseases. Write a Python program to process and analyze the DNA sequences by performing various operations based on nucleotide sequence rules. The program should perform the following on valid DNA Sequences:

- Accept 'n' DNA sequences from the user.
- Verify whether each sequence contains only the valid DNA bases (A, C, G, T). If so, display as Valid otherwise invalid.
- Search for the specific "TATAAA" sequence within the given DNA sequences. If so Display as match/nomatch
- Count and display the occurrences of base 'A' in each DNA sequence.

• Store the unique bases and their frequency in a dictionary for all sequences combined and display them. • Replace a specific substring in the DNA sequences with 'X' (e.g. replacing "AT" in "TATA" results in "TXA"). • Convert DNA to RNA by replacing 'T' with 'U', and store the RNA sequences in a list. • Search for a specific pattern in the DNA sequences and display its position if found. • Compute and display the complementary strand for each DNA sequence, where A pairs with T, and C pairs with G (e.g., "ATCGA" → "TAGCT"). Input Format First line contains number of DNA sequences Next line contains the DNA sequence 1 Next line contains the DNA sequence 2 . Next line contains the DNA sequence n Next line contains the specific sequence Next line contains the base to find its occurrence: Next line contains the substring to replace: Next line contains the string to replace with: Next line contains the string to replace to convert DNA to RNA Next line contains the string to replace with to convert DNA to RNA Next line contains the pattern to display its position Output Format First Line displays DNA sequence 1 valid/invalid Next Line displays DNA sequence 2 valid/invalid . Next Line displays DNA sequence n valid/invalid Next Line displays DNA sequence 1 Match/No Match Next Line displays DNA sequence 2 Match/No Match . Next Line displays DNA sequence n : Match/No Match Next Line displays DNA sequence 1 count Next Line displays DNA sequence 2 count . Next Line displays DNA sequence n count Frequency of Unique bases: {'A': count, 'T': count, 'G': count, 'C': count} Next Line displays DNA sequence 1: New DNA sequence 1 after replacing Next Line displays DNA sequence 2: New DNA sequence 2 after replacing . Next Line displays DNA sequence n: New DNA sequence n after replacing DNA to RNA:['New Sequenc1', 'New Sequenc2',… 'New Sequencn'] Next Line displays the position of the given Pattern in DNA sequence 1:	

Next Line displays the position of the given Pattern in DNA sequence 2: . Next Line displays the position of the given Pattern in DNA sequence n: Display the position of the given Pattern in DNA sequence 1: Display the position of the given Pattern in DNA sequence 2: Display the position of the given Pattern in DNA sequence n: Display the complementary strand of DNA sequence 1: Display the complementary strand of DNA sequence 2: . Display the complementary strand of DNA sequence n:	
6. **Renewable Energy Grid** In the year 2050, the world is powered by solar energy storage units, distributed across three major smart power stations. These stations are: (a) Station 1: Primary source, (b) Station 2: Intermediate (buffer) , (c) Station 3: Final destination. To ensure sustainable and stable energy flow, scientists have developed a precise transfer protocol that moves energy units from Station 1 to Station 3. Transfer Rules (Constraints): • Only one energy unit can be transferred at a time to prevent system overload. • A larger unit can never be placed on top of a smaller unit. • The number of energy units n must be an integer between 1 and 20. • The transfer output must be a sequence of moves: each move is a pair (a, b) meaning "move a unit from Station a to Station b", where a and b are in {1, 2, 3}. • Two consecutive moves must not involve the same pair of stations in opposite directions. For example, (1, 3), (3, 1) back-to-back is not allowed, as it causes unnecessary energy loss and instability. Input Format: An integer n represents the number of energy storage units to be transferred from the primary station to the final station. Output Format: A list of ordered pairs (a,b), where each pair represents a transfer of an energy unit from station a to station b. **Rainfall prediction using Sorting** Write a Python program to assist meteorologists in forecasting rainfall by analyzing relative humidity based on air temperature (23°C to 28°C) and dew point temperature (7°C to 16°C). Use the given formulas to compute specific humidity, saturation point, and relative humidity: specific humidity = 6.11 x 10^{7.5 x dew point} / {237.3 + dew point} saturation point = 6.11 x 10^{7.5 x air temp} / {237.3 + air temp}	4 hours

relative humidity = (specific humidity) / (saturation point) x 100 The chance of rainfall is high when the relative humidity is high and the temperature is dropping. The program should sort the relative humidity values in increasing order and display the highest relative humidity along with its corresponding air temperature and dew point temperature, providing valuable insights for rainfall prediction. Input Format: An integer 'N' representing the number of temperature readings. 'N' lines, each containing: Air Temperature (integer, between 23°C and 28°C) Dew Point Temperature (integer, between 7°C and 16°C) Output Format: Sorted list of relative humidity values in increasing order along with their corresponding air temperature and dew point temperature. Highest relative humidity value with its associated air temperature and dew point temperature. Prediction statement indicating whether the chance of rainfall is High/Low (if humidity is high and temperature is dropping).	
7. **Browser Tracking** Write a Python program to simulate a web browser's back and forward navigation system using stacks, where visiting a new webpage pushes the URL onto the back stack and clears the forward stack. Implement functions to navigate back by popping the top page from the back stack and pushing it onto the forward stack, and to navigate forward by popping the top page from the forward stack and pushing it onto the back stack. The program should allow users to enter URLs, navigate back and forward, and display the current page along with available navigation options dynamically. Input Format: •An integer N representing the number of webpage visits. •N lines, each containing a URL (string) representing a webpage visit. •User can enter the following commands dynamically after initial webpage visits: o BACK: Navigate to the previous page. o FORWARD: Navigate to the next page. o VISIT : Visit a new webpage, clearing the forward history. o EXIT: Terminate the program. Output Format: •Display the current page after each operation. •Show available navigation options (Back and Forward stacks). •Display messages when back or forward navigation is unavailable.	6 hours

•Display Browser session ended when Exit. **Customer Orders Queue Processing** In a smart manufacturing unit, automated machines process customer orders in a First-In-First-Out (FIFO) manner to ensure efficiency. Write a Python program using a queue to simulate an order processing system, where customers place orders for different industrial products (e.g., raw materials, electronic components, machinery parts), and each order is stored along with its order placed time. The system should process the orders in the same sequence they were received, record the order completion time, and remove them from the queue upon completion. Finally, display the status of the order queue before and after processing, along with the time taken to complete each order. Input Format: An integer 'N' representing the number of orders N lines, each containing: Order ID (string) Product Name (string) Order Placed Time (HH:MM 24hr -format) Order Completion Time ((HH:MM format) Output Format: Order queue before processing Processing details, including: Order ID - Product Name - Order Placed Time - Order Completion Time - Time Taken to Process (in minutes) Order queue after processing	
8. **Smart Building** Elena Vasquez, a futuristic architect, is designing modular smart buildings on a blueprint. Each building is represented as a rectangle with: •(x, y) — coordinates of the bottom-left corner •Width and height — horizontal and vertical sizes of the building To avoid overlap during construction, Elena needs a tool that calculates the exact overlapping area between any two buildings. Write a python function (collections framework) that takes two building rectangles as input and returns the area of overlap. If they do not overlap, return 0. Input Format: Each rectangle is given as a list of 4 integers: [x, y, width, height]	6 hours

Output Format:
A single integer — the overlapping area between the two buildings. If no overlap, return 0.
Constraints:
•All coordinates and dimensions are integers.
•Width and height must be positive (greater than 0).
•Input lists must contain exactly 4 integers.
•Coordinates may be negative (to allow buildings in all quadrants).

Analyze Water Quality
Environmental agencies, researchers assess the overall water quality, monitor potential pollution sources, and ensure the water is safe for intended uses like drinking, agriculture, and aquatic life using water pH and dissolved oxygen levels and turbidity. Assist them by developing a Python program using functions and conditional statements to analyse the water quality based on pH, turbidity, and dissolved oxygen levels. The program should take input values for these parameters, check if they are within safe limits, and classify the water as safe or unsafe for consumption.
If pH value is 6.5 and 8.5, Turbidity lesser than 3 NTU(Nephelometric Turbidity Unit) and Dissolved Oxygen between 6.5 and 8 mg/L then the water quality is safe Use conditional statements to give recommendations if the water is safe or unsafe. Allow users to input data for any location and display a summary of safe and unsafe water sources. Locations may include Clean River area, lake near agriculture area, urban storm drain area, Deep Ground water Source etc
Use menu driven approach and functions to compute and display for multiple locations

Input Format
First Line contains the name of the Location 1
Second Line contains the pH value
Third Line contains the Turbidity
Fourth Line contains Dissolved Oxygen
Enter either Yes/No
.
Next Line contains the name of the Location ‘n’
Next Line contains the pH value
Next Line contains the Turbidity
Next Line contains Dissolved Oxygen
Enter either Yes/No

Output Format
Safe/unsafe
Do you want to continue (Yes/No). If Yes
.
Safe/unsafe
Do you want to continue (Yes/No). If No, display ‘Thank you’

9. **Reforestation** In a reforestation project, each tree grows a fixed number of saplings every year. These saplings become mature trees starting from the next year and also begin producing saplings. Implement a python program using functions that calculates the total number of mature trees after a given number of years. Input Format: A tuple of three integers: initial_trees – Number of trees at year 0 saplings_per_tree – Number of saplings each tree produces per year years – Number of years to simulate Output Format: A single integer: total number of mature trees after the given number of years. **Recurring Character** For a given string which is made of uppercase letters (A–Z). Implement a recursive function that finds the first character that appears more than once. The function should return a dictionary with: • The character (key) • A list of two indices: • The index where it first appeared • The index where it reappeared first • If no recurring character is found, or if the input is None or an empty string, return an empty dictionary {}. Rules & Constraints: • Input must be a string containing only uppercase A–Z. • Return the first recurring character based on position. • Ignore later repetitions after the first recurrence. • Use recursion (no loops allowed). • Return an empty dictionary if: • Input is None • Input is an empty string • No character repeats Input Format: A single string: "ABCDAB" Output Format: A dictionary like: {"A": [0, 4]} Or: {} (if no recurring character or invalid input)	6 hours

Tower of Hanoi In robotic warehouse automation, autonomous robotic arms move stacked containers or pallets between storage locations while following strict stacking rules. Similar to the Tower of Hanoi problem, the system must transfer stacked items from a source rack to a destination rack using an intermediate rack, ensuring that larger items are never placed on smaller ones. Write a Python program using a recursive function to simulate this process, where a set number of stacked containers (disks) must be moved while following the First-In-Last-Out (FILO) principle. The program should determine the optimal sequence of moves, ensuring safe and efficient stacking, and display each step of the movement process to track how robotic arms relocate items. Input Format: An integer 'N' representing the number of stacked containers. Output Format: A sequence of steps showing how the containers are moved from the Source Rack to the Destination Rack using the Intermediate Rack.	
10. **Student Attendance Data Analysis using Numpy** Involve as a team to collect attendance dataset from professors. Design a Python program using NumPy to analyse a student attendance dataset. Compute the total attendance per student, identify the student with the highest attendance, calculate the class average attendance per day, and determine the day with the lowest attendance rate. Finally, compute each student's attendance percentage and display students who have less than 75% attendance. Input Format: •An integer N representing the number of students. •An integer D representing the number of days. •A N x D matrix, where each row corresponds to a student, and each column represents a day. o1 indicates present o0 indicates absent Output Format: •Display the original attendance matrix. •Compute and display the total attendance per student. •Identify and display the student with the highest attendance. •Calculate and display the class average attendance per day. •Determine and display the day with the lowest attendance rate. •Compute and display each student's attendance percentage. •Identify and display students who have less than 75% attendance.	4 hours

11. **Student Marks Data Analysis using Pandas** Involve as a team to collect sample archived student marks dataset from professors. Write a Python program using Pandas to analyse students' internal marks, where each student is evaluated based on six internal components and a final component. Create a DataFrame with student names, six internal scores, and final score. Calculate the class average internal score, compute the total marks for each student by summing all components, and determine the student with the highest total marks. Additionally, filter and display students who have scored below 40% indicating the need for improvement. Input Format: An integer 'N' representing the number of students. N lines, each containing: Student Name (String) Six Internal Component Scores (Integers from 0 to 100) Final Component Score (Integer from 0 to 100) Output Format: •Display the original DataFrame with student names, internal scores, and final scores. •Compute and display the class average internal score. •Calculate and display the total marks for each student. •Identify and display the student with the highest total marks. •Filter and display students scoring below 40% of the total (i.e., below 40 out of 100).	4 hours
12. **Restaurant Food Waste Cost** Restaurants throw away a lot of food every day, which affects both costs and the environment. Consider the dataset which contains details of food waste over a month. Write a python program using Pandas to analyze and reduce this waste. Input Format: A CSV file with the following columns: •Date (YYYY-MM-DD) – The day waste was recorded •Food Item – Name of the food item wasted •Category – Type of food (e.g., Vegetables, Dairy, Meat) •Quantity Wasted – Amount wasted (in kg or units) •Cost per Item – Cost of one unit of the food item Perform the following: 1. Clean the Data o Handle missing values in Date, Food Item, Category, Quantity Wasted, or Cost per Item (e.g., fill or remove).	6 hours

2. Calculate Waste Cost
   o Per Day: Total waste cost for each day
   o Per Category: Waste cost for each food category
3. Top 5 Most Wasted Items
   o Find the 5 food items with the highest total quantity wasted
   o Suggest simple strategies to reduce waste for each (e.g., store better, reuse, order less)
4. Trend & Strategy Analysis
   o Spot patterns (e.g., which days waste the most)
   o Recommend ways to reduce waste (e.g., smaller portions, better planning, donation)

Output Format:
•A cleaned DataFrame (after handling missing data)
•Two summary tables:
 o Total waste cost per day
 o Total waste cost per category
•A ranked list of top 5 wasted food items, with suggestions
•Trend insights and recommendations for reducing waste.

Smart Energy Grid
A smart energy grid tracks hourly electricity usage (in kWh) for a residential building over 30 days. The data is stored in a NumPy array. Write a Python program that analyzes the energy usage using NumPy.
Input Format:
A NumPy array with shape (30, 24):
30 rows = 30 days
24 columns = 24 hours per day
Each value shows the electricity used during that hour.
Perform the following:
1. Calculate Energy Usage
  • Daily Consumption: Total energy used per day (output: array of 30 values)
  • Monthly Consumption: Total energy used in 30 days (output: one float)
2. Find Peak Hours
For each day, find hours where usage is above that day's average
Output: A dictionary {day_index: [list_of_peak_hours]}
3. Predict Next Day's Usage
Use the last 7 days to predict the next day's total energy usage using simple average of daily totals from the last 7 days

Output Format:
Daily Consumption: NumPy array (30 values)
Monthly Consumption: Float
Peak Hours: Dictionary of peak hour indices per day
Predicted Next-Day Usage: Float (average of last 7 days)

Total Laboratory Hours:	**60 hours**
Text Book(s)	
John V. Guttag, "**Introduction to computation and programming using Python** ", The MIT Press, 3rd Edition, **2021** Eric Matthes , "**Python Crash Course.** ", No Starch Press, 3rd Edition, **2022**	
Reference Books	
Luciano Ramalho, "**Fluent Python** ", OReilly Media Inc , 2nd Edition, **2022** Charles R. Severance , "**Python for Everybody** ", Shroff Publishers, 1st Edition, **2024**	
Mode of Evaluation : Lab Continuous Assessment, Lab Final Assessment	
Recommended by Board of Studies :	21-05-2025
Approved by Academic Council : No. 78	12-06-2025