K. K. Wagh Institute of Engineering Education & Research, Nashik

(An Autonomous Institute From A.Y. 2022-23)

WINTER-2025	
Exam Seat No.:	
Academic Year:2025-2026	Semester:V
Class:TY	Program:B.Tech
Branch Code:ROB	Pattern:2023
Name of Course:Python programming	Course Code:2312306A
Max. Marks:60	Duration:2.30 Hrs.

Instructions: Candidates should read carefully the instructions printed on the Question Paper and on the cover page of the Answer Book, which is provided for their use.

1. This question paper contains 4 pages.
2. Answer to each new question is to be started on a new page.
3. Assume suitable data wherever required, but justify it.
4. Draw the neat labelled diagrams, wherever necessary.
5. The last columns indicates the Course Outcome and level of Blooms Taxonomy of the Question/sub-question.

		Marks	**CO**
Question No. 1			
1a)	A robotic arm lifts an object using a hydraulic cylinder. Given the piston radius (in cm) and the applied pressure (in N/cm²), Write a Python program to: i. Input piston radius and pressure. ii. Calculate force = π × radius² × pressure. iii. Display radius, pressure, and force using f-string formatting (3 decimals).	(6)	CO1
Question No. 2			
2a)	In a robotic application, you want to visualise the stacking of boxes in a triangular shape. Write a Python program that displays a pattern of stars to represent the stack, as shown below: * ** *** **** *****	(6)	CO2
Question No. 3			
3a)	Write a Python function that calculates the productivity of a robotic assembly line based on the number of units assembled, the total operating hours, and a speed factor (a multiplier representing robot speed). Return the productivity in units per hour. Use the formula: productivity = (units_assembled / operating_hours) × speed_factor Test your function with the following input values: i. units_assembled = 120, operating_hours = 8, speed_factor = 1.1 ii. units_assembled = 200, operating_hours = 10, speed_factor = 0.95	(8)	CO3

The function should handle invalid inputs (e.g., zero or negative values) by returning an appropriate error message.

OR

3b) Write a Python function calculate_arm_stress that computes the stress experienced by a robotic arm link based on the applied load, cross-sectional area, and a safety factor. (8) CO3

Use the following formula:

$$\text{stress} = \frac{\frac{\text{load}}{\text{area}}}{1 + (k \times \text{area})}$$

where:

- load is in Newtons
- area is in square centimeters
- k is the safety factor coefficient

The function should return the stress value in N/cm² formatted to 2 decimal places.

Test your function with:
i. load = 500, area = 25, k = 0.02
ii. load = 800, area = 40, k = 0.03

Handle invalid inputs (e.g., zero or negative values) by returning an appropriate error message.

3c) Create a Python package called robotics that includes a module named power.py. This module should have a function to calculate the mechanical power output of a robotic motor. Use the formula: (8) CO3
power = torque × angular_velocity

Steps to implement and test:

1. In the power.py module inside the robotics package, define a function calculate_power(torque, angular_velocity) that takes the torque (in N·m) and angular velocity (in rad/s) as parameters and returns the mechanical power (in Watts).
2. Write a separate script to:
   i. Import the calculate_power function from robotics.power.
   ii. Set torque = 10 and angular_velocity = 25.
   iii. Call the calculate_power function with these values and print the result, formatted to two decimal places.

OR

3d) Create a Python package named iot_monitoring that includes a module called light_sensor.py. This module should contain a function to convert light intensity from lumens to lux using the formula: (8) CO3

lux = lumens / area

Steps:

1. In light_sensor.py, define a function called lumens_to_lux that takes lumens and area (in square meters) as arguments and returns the light intensity in lux.
2. Write a script that:
   i. Imports the lumens_to_lux function from the iot_monitoring.light_sensor module.
   ii. Consider lumens = 1500 and area = 10.
   iii. Calls the function with these values and prints the resulting lux value, formatted to two decimal places.

Question No. 4

4a) Write a Python program with a function that processes a command for a robotic assistant. Follow the steps below to implement and test the function: (8) CO4

1. Define a function named process_robotic_command() that takes a single command as input.
2. Inside the function:

i. Remove any extra spaces at the beginning and end of the command.
ii. Convert the entire command to lowercase.

iii. Replace occurrences of the word "light" with "LED."
iv. Split the command into individual words.
v. Display the final command in uppercase to confirm the changes.
3. Call process_robotic_command("Turn on the Light") to test the function and demonstrate each modification step.
Each step in the function should print the updated command to show how it changes at every stage.

OR

4b) Write a Python program for a home-assistant robot that checks environmental control commands. (8) CO4
1. Define a function check_environment_command(command).
2. Inside the function:
i. Use the "in" operator to check if words like "heat" or "cool" appear (to identify temperature control actions).
ii. Use the "not in" operator to ensure that words like "fault" or "disconnect" are absent (indicating system safety).
iii. Print messages indicating whether the robot can execute the command safely or must reject it.
3. Test the function with commands such as:
- "heat the room"
- "cool the environment"
- "disconnect cooling system"
- "heat mode fault detected"

4c) You are working with sensor data for a robot, and you have the following lists: (8) CO4
- sensors = ['temperature', 'humidity', 'battery', 'motion']
- values = [75, 40, 85, True]

Tasks:
1. Use dictionary comprehension to create a dictionary called sensor_status where the keys are from the sensors list and the values are from the values list.
2. Print the status of the battery.
3. Update the humidity to 50 and print the updated dictionary
4. Add a new sensor status for "light" with a value of 60 and print the updated dictionary.
5. Check if the key "motion" is present in the dictionary and print an appropriate message.

OR

4d) In an industrial robot system, we need to store and manage the operational status of different actuators. Create a dictionary to hold the following actuator details: (8) CO4
"motor_speed": 1500 (in rpm)
"arm_position": 45 (in degrees)
"gripper": "closed"
"camera_active": True (indicates if the camera is on)
Perform the following tasks:
1. Print the current motor speed.
2. Update the arm_position to 90 degrees and print the updated status.
3. Add a new actuator "conveyor_belt" with a value of "running" and print the updated dictionary.
4. Check if the key "camera_active" exists in the dictionary and print an appropriate message.

Question No. 5

5a) You need to implement a BatteryRobot class with the following functionalities: (8) CO5
i. The class should have two attributes, battery1 and battery2, representing the battery levels (in percentage) of two different power cells, initialized through the constructor.
ii. Implement a method total_battery(self) that returns the combined battery percentage of both cells.
iii. Implement a method battery_difference(self) that returns the difference between the battery levels of battery1 and battery2.
iv. Implement a method average_battery(self) that calculates and returns the average battery percentage of both cells.
v. Implement a method battery_status(self) that checks if either battery level is critically low (below 20%) and returns a warning message. Otherwise, it returns "Battery levels are sufficient."
vi. Implement a method display_status(self) that prints the battery levels along with the results of the calculations and status check.
Write the complete code for the BatteryRobot class based on the specifications above, and demonstrate its usage with example battery levels (e.g., battery1 = 45.0 and battery2 = 18.0).

OR

5b) You need to implement a SensorRobot class with the following functionalities: (8) CO5

i. Sensor Initialization: The class should have attributes sensor1 and sensor2 to represent the readings from two sensors (e.g., distance measurements), which are initialized through the constructor.

ii. Addition Method: Implement a method add_sensors(self) that returns the sum of the sensor readings.

iii. Subtraction Method: Implement a method subtract_sensors(self) that returns the difference between the first sensor reading and the second sensor reading.

iv. Multiplication Method: Implement a method multiply_sensors(self) that returns the product of the two sensor readings.

v. Division Method: Implement a method divide_sensors(self) that returns the result of dividing the first sensor reading by the second. Ensure to handle the case where the second sensor reading is zero, returning an appropriate message.

vi. Display Method: Implement a method display_results(self) that prints the sensor readings along with the results of the arithmetic operations.

Write the complete code for the SensorRobot class based on the specifications above, and demonstrate its usage with example sensor readings (e.g., 12.5 and 3.0).

5c) Design a class hierarchy for inspection drones using inheritance. (8) CO5

1.Base Class: InspectionDrone

i. Attributes: drone_id (unique identifier), drone_type (type of drone, e.g., "Camera" or "Thermal").

ii. Method: drone_info() which prints the drone's ID and type.

2.Derived Classes:

i. CameraDrone: Inherits from InspectionDrone

ii. ThermalDrone: Inherits from InspectionDrone

Create instances of CameraDrone and ThermalDrone. Call the drone_info() method on both objects.

OR

5d) Design a class hierarchy to represent different types of smart home devices using inheritance. (8) CO5

1. Base Class: SmartDevice

i. Attributes: device_name (name of the device) and device_type (e.g., "Light" or "Thermostat").

ii. Method: device_info(), which prints the device's name, type, and status.

2. Derived Classes:

i. SmartLight: Inherits from SmartDevice

- Additional Attribute: brightness (default is 0).

ii. SmartThermostat: Inherits from SmartDevice

- Additional Attribute: temperature (default is 20).

Tasks:

1. Create instances of SmartLight and SmartThermostat.
2. Call the device_info() method on both objects.

····· End of question paper·····