Roll Number: ______________________	
Thapar Institute of engineering and Technology, Patiala Department of Mechanical Engineering	
B.E. Robotics and Artificial Intelligence (2nd Year) MST	**URA302: Python Programming**
29th Sept. 2025; 11:30 AM Time: 2 Hours; MM: 30	Name of Faculty: Dr. Rohit Kr. Singla

Note: Attempt all questions. Take suitable data wherever necessary.

S. No.	Question	Marks	CO	BL
Q1	Write a Python program that takes two numbers `x = 25` and `y = 7`, computes a result using integer division and remainder, checks a Boolean condition, and manipulates strings as follows: • First, calculate `num_result` as the sum of the integer division of `x` by `y` and the remainder when `x` is divided by `y`. • Then, create a Boolean variable `bool_result` that is `True` if `num_result` is greater than 5 and `y` is less than 10. • Next, join the strings `"Robotics"` and `"AI"` into a single string `s`. • Finally, construct a new string `final_string` by taking the first 5 characters of `s`, adding `num_result` (converted to a string), and then appending the last 2 characters of `s`. Print all four results: `num_result`, `bool_result`, `s`, and `final_string`.	(6)	CO1	L2
Q2	Explain Python containers with suitable examples: • List (creation, slicing, and iteration using loops) • Tuple (use-case) • Dictionary (key–value access) • Set (operations union and intersection) 4 Also compare list vs tuple and set vs dictionary briefly. 2	(6)	CO1	L2
Q3(a)	Object-Oriented Programming (OOP) is based on four main principles: Encapsulation, Inheritance, Polymorphism, and Abstraction. (a) Define each principle in your own words. (b) For each principle, write a short Python example that demonstrates its use. (c) Explain why these principles make Python programs more structured and reusable.	(3)	CO1	L3
Q3(b)	In Python, functions and classes are widely used in Robotics and AI to structure code for reusability and modularity. Write a Python program that defines a class Robot with attributes like name and battery_level, and a method to display the robot's status. 2 Also, create a function outside the class that takes two numbers (e.g., sensor readings) and returns their average. 1	(3)		

① Demonstrate the use of both by creating an object of the class and calling the function.

Q4 You are working on sensor data for a mobile robot. The readings from three sensors (front, left, right) over three time steps are stored in a NumPy array:

```
import numpy as np
data = np.array([[12.5,  9.7, 14.2],
                 [10.1, 11.5, 13.3],
                 [ 9.8, 12.0, 15.4]])
```

Tasks:

1. (Array Indexing) Extract the front sensor reading at the 2nd time step.
2. (Slicing) Get all readings from the left sensor as a 1D array.
3. (Data Types) Convert the array `data` to integers.
4. (Array Math) Subtract `10` from all readings using broadcasting.
5. (Aggregation) Compute the average reading of each sensor (column-wise mean).
6. (Advanced Indexing) Find the maximum value and its position (row, column).

6 (6) CO3 L4

Q5 A mobile robot records its distance travelled (in meters) at different time intervals (in seconds) as follows:

Time (s)	0	2	4	6	8
Distance (m)	0	4	8	15	20

Tasks:

1. Using SciPy, perform a cubic interpolation to estimate the robot's distance at t = 5 s. 2
2. Plot the original points and the interpolated curve using Matplotlib (Just write code for the same). 2
3. Using manual cubic interpolation formula, approximate the distance at t = 5 s using the nearest 4 points (Lagrange interpolation). Show all steps. 1
4. Discuss briefly which interpolation method (linear vs cubic) is better for smooth robotic motion prediction and why. 1

(6) CO3 L4

***** All the Best *****

Roll Number:______________________

Thapar Institute of engineering and Technology, Patiala
Department of Mechanical Engineering

B.E. Robotics and Artificial Intelligence (2nd Year) MST	**URA302: Python Programming**
29th Sept. 2025; 10:30 AM Time: 2 Hours; MM: 30	Name of Faculty: RKS

Note: Attempt all questions. Take suitable data wherever necessary.

S. No.		Question	Marks	CO	BL
Q1	a)	Write a Python program that takes two numbers `x = 25` and `y = 7`, computes a result using integer division and remainder, checks a Boolean condition, and manipulates strings as follows: • First, calculate `num_result` as the sum of the integer division of `x` by `y` and the remainder when `x` is divided by `y`. • Then, create a Boolean variable `bool_result` that is `True` if `num_result` is greater than 5 **and** `y` is less than 10. • Next, join the strings `"Robotics"` and `"AI"` into a single string `s`. • Finally, construct a new string `final_string` by taking the first 5 characters of `s`, adding `num_result` (converted to a string), and then appending the last 2 characters of `s`. Print all four results: `num_result`, `bool_result`, `s`, and `final_string`.	(12) 6	CO1	L1

Step 1: Define the numbers

```
x = 25
y = 7
```

Step 2: Compute `num_result`

- Integer division: `x // y = 25 // 7 = 3`
- Remainder: `x % y = 25 % 7 = 4`
- Sum: `3 + 4 = 7`

So:

```
num_result = (x // y) + (x % y)   # 7
```

Step 3: Boolean check

- `num_result > 5 → 7 > 5 → True`
- `y < 10 → 7 < 10 → True`
- Combined → `True and True → True`

```
bool_result = (num_result > 5) and (y < 10)   # True
```

Step 4: String join

```
s = "Robotics" + "AI"   # "RoboticsAI"
```

Step 5: Construct `final_string`

- First 5 characters of `s`: `"Robot"`
- `num_result` as string: `"7"`
- Last 2 characters of `s`: `"AI"`

So result:

```
final_string = s[:5] + str(num_result) + s[-2:]    #
"Robot7AI"
```

✓ Full Python Program

```
x = 25
y = 7

# Step 1: Compute num_result
num_result = (x // y) + (x % y)

# Step 2: Boolean check
bool_result = (num_result > 5) and (y < 10)

# Step 3: Join strings
s = "Robotics" + "AI"

# Step 4: Construct final_string
final_string = s[:5] + str(num_result) + s[-2:]

# Print results
print("num_result:", num_result)
print("bool_result:", bool_result)
print("s:", s)
print("final_string:", final_string)
```

1 1 1 1 1 1

🎯 Output:

```
num_result: 7
bool_result: True
s: RoboticsAI
final_string: Robot7AI
```

Q2	~~a)~~	Explain Python containers with suitable examples: • List (creation, slicing, and iteration using loops) • Tuple (immutability and use-case) • Dictionary (key–value access) • Set (uniqueness and operations like union/intersection) 4 Also compare list vs tuple and set vs dictionary briefly. 2	(12) 6	CO1	L2

Python Containers with Examples

Python provides several container data types to store and manage collections of data. The most common ones are List, Tuple, Dictionary, and Set.

---

1. List

- A list is an ordered, mutable (changeable) collection of items.
- Elements can be of different data types.

✓ Example: Creation, slicing, and iteration

```
# Creation
fruits = ["apple", "banana", "cherry", "date",
"elderberry"]

# Slicing
print(fruits[1:4])    # ['banana', 'cherry', 'date']
print(fruits[-2:])    # ['date', 'elderberry']

# Iteration using loop
for fruit in fruits:
    print(fruit)
```

◆ Output:

```
['banana', 'cherry', 'date']
['date', 'elderberry']
apple
banana
cherry
date
elderberry
```

2. Tuple

- A tuple is similar to a list, but it is immutable (cannot be modified after creation).
- Used when you want read-only data that should not be accidentally changed.

✅ Example: Immutability & Use-case

```
# Creation
coordinates = (10.5, 20.8)

print("X:", coordinates[0])
print("Y:", coordinates[1])

# Trying to modify → will give error
# coordinates[0] = 15.0    # ✖ TypeError
```

◆ Use-case: Tuples are great for storing fixed data like geographic coordinates, RGB color values, etc., where immutability is desirable.

3. Dictionary

- A dictionary is a collection of key–value pairs.
- Keys must be unique and immutable (like strings, numbers, tuples).
- Values can be any type.

✅ Example: Key–value access

```
# Creation
student = {
    "name": "Alice",
    "age": 21,
    "course": "AI"
}

# Accessing values
print(student["name"])    # Alice
print(student.get("age"))  # 21

# Adding new key-value
student["grade"] = "A"
print(student)
```

◆ Output:

```
Alice
21
{'name': 'Alice', 'age': 21, 'course': 'AI', 'grade':
'A'}
```

4. Set

- A set is an unordered collection of unique items.
- No duplicates allowed.

		• Useful for mathematical set operations like union, intersection, etc.			

✅ Example: Uniqueness and operations

```
# Creation
set_a = {1, 2, 3, 4}
set_b = {3, 4, 5, 6}

# Uniqueness
numbers = {1, 2, 2, 3, 3, 4}
print(numbers)   # {1, 2, 3, 4}

# Union
print(set_a | set_b)    # {1, 2, 3, 4, 5, 6}
```

Set a. Union (Set2)

```
# Intersection
print(set_a & set_b)    # {3, 4}
```

Set a. intersection (Set2)

◆ Comparisons

List vs Tuple

Feature	List	Tuple
Mutability	Mutable (can change)	Immutable (cannot change)
Performance	Slightly slower	Faster (because fixed)
Use-case	Dynamic data (shopping cart, student records)	Fixed data (coordinates, constants)

Set vs Dictionary

Feature	Set	Dictionary
Structure	Collection of unique values	Collection of key–value pairs
Access	No indexing, unordered	Access by key (fast lookup)
Use-case	Membership test, mathematical set ops	Storing structured data (name–age, id–value mapping)

✅ Final Notes

- List → Ordered, mutable, good for general collections.
- Tuple → Ordered, immutable, good for fixed data.
- Dictionary → Key–value storage, fast lookups.
- Set → Unique, unordered, good for eliminating duplicates and set operations.

Q3	a)	Object-Oriented Programming (OOP) is based on four main principles: Encapsulation, Inheritance, Polymorphism, and Abstraction. (a) Define each principle in your own words.	3 (5)	CO3	L6

		(b) For each principle, write a short Python example that demonstrates its use. (c) Explain why these principles make Python programs more structured and reusable.			

(a) Definitions

1. Encapsulation → Bundling data (attributes) and methods (functions) into a single unit (class) and restricting direct access to protect data.
2. Inheritance → A class can inherit properties and methods from another class, promoting code reuse.
3. Polymorphism → The ability of objects/methods to take on different forms (e.g., same method name behaving differently for different classes).
4. Abstraction → Hiding implementation details and showing only the essential features using abstract classes or methods.

(b) Examples

1. Encapsulation

```
class BankAccount:
    def __init__(self, balance):
        self.__balance = balance    # private attribute

    def deposit(self, amount):
        self.__balance += amount

    def get_balance(self):
        return self.__balance

account = BankAccount(1000)
account.deposit(500)
print(account.get_balance())  # 1500
```

2. Inheritance

```
class Animal:
    def sound(self):
        print("Some sound")

class Dog(Animal):  # inherits from Animal
    def sound(self):
        print("Bark")

dog = Dog()
dog.sound()  # Bark
```

3. Polymorphism

```
class Cat:
    def sound(self):
        return "Meow"

class Dog:
    def sound(self):
        return "Bark"

for animal in [Cat(), Dog()]:
    print(animal.sound())
```

◆ Output:

```
Meow
```

```
Bark
```

4. Abstraction

```
from abc import ABC, abstractmethod

class Shape(ABC):
    @abstractmethod
    def area(self):
        pass

class Circle(Shape):
    def __init__(self, r):
        self.r = r
    def area(self):
        return 3.14 * self.r * self.r

circle = Circle(5)
print(circle.area())  # 78.5
```

(c) Importance

- Encapsulation → Protects data and prevents misuse.
- Inheritance → Promotes reusability and reduces redundancy.
- Polymorphism → Increases flexibility by allowing different objects to be treated uniformly.

Abstraction → Simplifies complexity by exposing only essential details.

b) In Python, functions and classes are widely used in Robotics and AI to structure code for reusability and modularity. Write a Python program that defines a class Robot with attributes like name and battery_level, and a method to display the robot's status. Also, create a function outside the class that takes two numbers (e.g., sensor readings) and returns their average. Demonstrate the use of both by creating an object of the class and calling the function. (10) 3

```
# Function outside the class
def calculate_average(sensor1, sensor2):
    """
    Function to calculate the average of two sensor
readings.
    """
    return (sensor1 + sensor2) / 2
```

1

```
# Defining the Robot class
class Robot:
    def __init__(self, name, battery_level):
        # Attributes
        self.name = name
        self.battery_level = battery_level

    # Method to display robot's status
    def display_status(self):
        print(f"□ Robot Name: {self.name}")
        print(f"■ Battery Level:
{self.battery_level}%")
```

1

```
# --- Demonstration ---
```

```
# Create an object of Robot class
robot1 = Robot("Atlas", 85)

# Call method inside class
robot1.display_status()

# Use function outside the class
sensor_reading1 = 40
sensor_reading2 = 50
avg_reading = calculate_average(sensor_reading1,
sensor_reading2)

print(f"\n📡 Average of sensor readings
({sensor_reading1}, {sensor_reading2}) =
{avg_reading}")
```

1

✓ Expected Output

```
□ Robot Name: Atlas
🔋 Battery Level: 85%

📡 Average of sensor readings (40, 50) = 45.0
```

◆ Explanation:

1. The `Robot` class groups related attributes (`name`, `battery_level`) and a method (`display_status`).
2. The function `calculate_average` is defined outside the class for modularity and reusability.
3. We created a `Robot` object (`robot1`) and demonstrated OOP principles.

We also used the function to process sensor data separately.

Q4

You are working on sensor data for a mobile robot. The readings from three sensors (front, left, right) over three time steps are stored in a NumPy array:

```
import numpy as np
data = np.array([[12.5,  9.7, 14.2],
                 [10.1, 11.5, 13.3],
                 [ 9.8, 12.0, 15.4]])
```

Tasks:

1. (Array Indexing) Extract the front sensor reading at the 2nd time step.
2. (Slicing) Get all readings from the left sensor as a 1D array.
3. (Data Types) Convert the array `data` to integers.
4. (Array Math) Subtract `10` from all readings using broadcasting.
5. (Aggregation) Compute the average reading of each sensor (column-wise mean).
6. (Advanced Indexing) Find the maximum value and its position (row, column).

~~(12)~~ 6 | CO3 | L3

```python
import numpy as np

data = np.array([[12.5,  9.7, 14.2],
                 [10.1, 11.5, 13.3],
                 [ 9.8, 12.0, 15.4]])
```

1. Array Indexing

Extract the front sensor reading at the 2nd time step.

- Convention: rows → time steps, columns → sensors (`0=front, 1=left, 2=right`).
- 2nd time step = row index `1` (0-based).
- Front sensor = column index `0`.

```python
front_second = data[1, 0]
print(front_second)   # 10.1
```

✅ Output:

```
10.1
```

2. Slicing

Get all readings from the left sensor as a 1D array.

- Left sensor = column index `1`.

```python
left_sensor = data[:, 1]
print(left_sensor)   # [ 9.7 11.5 12.0]
```

✅ Output:

```
[ 9.7 11.5 12. ]
```

3. Data Types

Convert the array to integers.

```python
int_data = data.astype(int)
print(int_data)
```

✅ Output:

```
[[12  9 14]
 [10 11 13]
 [ 9 12 15]]
```

4. Array Math (Broadcasting)

Subtract `10` from all readings.

```python
subtracted = data - 10
print(subtracted)
```

✅ Output:

```
[[ 2.5 -0.3  4.2]
 [ 0.1  1.5  3.3]
 [-0.2  2.0  5.4]]
```

5. Aggregation

Compute the average reading of each sensor (column-wise mean).

```python
avg_sensor = np.mean(data, axis=0)
print(avg_sensor)
```

✅ Output:

```
[10.8 11.1 14.3]
```

6. Advanced Indexing

Find the maximum value and its position (row, column).

```python
max_value = np.max(data)
```

```
position = np.unravel_index(np.argmax(data),
data.shape)
print(max_value, position)
```

✅ Output:

```
15.4 (2, 2)
```

☞ Max = `15.4` at row `2`, column `2` (3rd time step, right sensor).

🎯 Final Answer Summary

1. `10.1`
2. `[ 9.7 11.5 12. ]`
3. Integer array →
4. `[[12  9 14]`
5. ` [10 11 13]`
6. ` [ 9 12 15]]`
7. Subtracted array →
8. `[[ 2.5 -0.3  4.2]`
9. ` [ 0.1  1.5  3.3]`
10. ` [-0.2  2.0  5.4]]`
11. Column-wise averages → `[10.8 11.1 14.3]`
12. Max value → `15.4` at `(2, 2)`

Q5

A mobile robot records its distance traveled (in meters) at different time intervals (in seconds) as follows:

Time (s)	0	2	4	6	8
Distance (m)	0	4	8	15	20

Tasks:

1. Using SciPy, perform a cubic interpolation to estimate the robot's distance at t = 5 s.
2. Plot the original points and the interpolated curve using Matplotlib.
3. Using manual cubic interpolation formula, approximate the distance at t = 5 s using the nearest 4 points. Show all steps.
4. Discuss briefly which interpolation method (linear vs cubic) is better for smooth robotic motion prediction and why.

6 (12) | CO3 | L3

Great robotics + math question 🚀 Let's carefully solve all four tasks.

We are given data:

Time (s)	0	2	4	6	8
Distance (m)	0	4	8	15	20

✅ Task 1: Cubic Interpolation with SciPy

We'll use `scipy.interpolate.interp1d` with `kind='cubic'`.

```
import numpy as np
from scipy.interpolate import interp1d
import matplotlib.pyplot as plt

# Given data
time = np.array([0, 2, 4, 6, 8])
distance = np.array([0, 4, 8, 15, 20])
```

```
# Create cubic interpolation function
cubic_interp = interp1d(time, distance, kind='cubic')

# Estimate at t = 5 s
t_new = 5
d_est = cubic_interp(t_new)
print(f"Estimated distance at t=5s (SciPy cubic): {d_est:.2f} m")
```

☞ Expected result:

```
Estimated distance at t=5s (SciPy cubic): ~11.48 m
```

✓ Task 2: Plot Original Points + Interpolated Curve

```
# Generate smooth curve
t_fine = np.linspace(0, 8, 100)
d_cubic = cubic_interp(t_fine)

# Plot
plt.scatter(time, distance, color='red', label="Original Points")
plt.plot(t_fine, d_cubic, label="Cubic Interpolation", color="blue")
plt.axvline(5, color='gray', linestyle='--', alpha=0.7)
plt.scatter([t_new], [d_est], color='green', label=f"t=5s: {d_est:.2f} m")

plt.xlabel("Time (s)")
plt.ylabel("Distance (m)")
plt.title("Robot Distance Interpolation")
plt.legend()
plt.grid(True)
plt.show()
```

This shows both the data points and the smooth cubic curve.

✓ Task 3: Manual Cubic Interpolation (Lagrange Polynomial)

We use Lagrange interpolation formula with 4 nearest points around `t = 5`.

Nearest 4 points: (2, 4), (4, 8), (6, 15), (8, 20).

The formula:

$$P(x) = \sum_{i=0}^{n} y_i \cdot L_i(x)$$

where

$$L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}$$

Step-by-step:

- Points:
  ((x_0, y_0) = (2, 4))
  ((x_1, y_1) = (4, 8))
  ((x_2, y_2) = (6, 15))
  ((x_3, y_3) = (8, 20))

Target: (x = 5).

Compute Lagrange basis terms at x=5:

$$L_0(5) = \frac{(5-4)(5-6)(5-8)}{(2-4)(2-6)(2-8)} = \frac{(1)(-1)(-3)}{(-2)(-4)(-6)} = \frac{3}{-48} = -0.0625$$

$$L_1(5) = \frac{(5-2)(5-6)(5-8)}{(4-2)(4-6)(4-8)} = \frac{(3)(-1)(-3)}{(2)(-2)(-4)} = \frac{9}{16} = 0.5625$$

$$L_2(5) = \frac{(5-2)(5-4)(5-8)}{(6-2)(6-4)(6-8)} = \frac{(3)(1)(-3)}{(4)(2)(-2)} = \frac{-9}{-16} = 0.5625$$

$$L_3(5) = \frac{(5-2)(5-4)(5-6)}{(8-2)(8-4)(8-6)} = \frac{(3)(1)(-1)}{(6)(4)(2)} = \frac{-3}{48} = -0.0625$$

Now construct polynomial:

$$P(5) = y_0L_0 + y_1L_1 + y_2L_2 + y_3L_3$$

$$= 4(-0.0625) + 8(0.5625) + 15(0.5625) + 20(-0.0625)$$

$$= -0.25 + 4.5 + 8.4375 - 1.25$$

$$= 11.4375$$

Manual cubic interpolation at `t=5s` ≈ 11.44 m (very close to SciPy result ~11.48 m .

✅ Task 4: Linear vs Cubic for Robotics

- Linear interpolation → Simple, connects two nearest points with straight lines.
  - Pros: Fast, easy.
  - Cons: Motion prediction can be jerky, not smooth (bad for robots).
- Cubic interpolation → Uses higher-order polynomials (smooth curves).
  - Pros: Produces smooth trajectories → better for robotic path planning and motion control.
  - Cons: Slightly more computation.

☞ Conclusion:
Cubic interpolation is better for robotic motion prediction since robots require smooth and continuous paths for stability and safety. Linear is only acceptable for quick, rough estimates.

***** All the Best *****