CONTENTS

LAB ACTIVITY 1

INTRODUCTION TO PYTHON, PYTHON DATA TYPES-I

1. **Temperature Conversion:** Write a Python program that takes user input for a temperature in Celsius and converts it to Fahrenheit. Display the result with an appropriate message.

2. **Character Counter:** Develop a Python script that takes a user input string and counts the number of occurrences of each character in the string. Display the results in a readable format.

3. **Even or Odd Checker:** Create a Python program that prompts the user to enter an integer and checks if it's an even or odd number. Display a message indicating whether the number is even or odd.

4. **Quadratic Equation Solver:** Write a Python script that solves a quadratic equation of the form $ax^2 + bx + c = 0$ given user input for coefficients a, b, and c. Display the solutions with appropriate messages.

5. **Factorial Calculator:** Develop a Python program that calculates the factorial of a given non-negative integer entered by the user. Display the result with a suitable message.

6. **Guess the Number Game:** Create a simple number-guessing game in Python. Generate a random number between 1 and 100, and allow the user to guess the number. Provide hints (higher or lower) until the correct number is guessed.

7. **List Operations:** Implement a Python program that performs the following operations on a list of numbers entered by the user:

   - Calculate the sum of the numbers.
   - Find the maximum and minimum values.
   - Sort the list in ascending order.

8. **Currency Converter:** Write a Python program that converts an amount in one currency to another. Allow the user to input the amount, original currency, and target currency. Display the converted amount.

LAB ACTIVITY 2

PYTHON DATA TYPES- II

1. **List Manipulation**
   (a) Write a Python program that takes a list of numbers as input and returns a new list containing only the even numbers.
   (b) Implement a function that reverses the elements of a given list in-place.
2. **Tuple Operations**
   (a) Create a Python script that accepts a tuple of strings, sorts them alphabetically, and prints the sorted tuple.
   (b) Write a function that finds the index of a specific element in a tuple.
3. **Dictionary Operations**
   (a) Develop a program that merges two dictionaries, where the keys are names, and the values are ages. If a key exists in both dictionaries, combine the ages.
   (b) Implement a function that checks if a given key exists in a dictionary.
4. **Set Operations**
   (a) Write a Python function that takes two sets as input and returns a new set containing the common elements.
   (b) Create a program that removes duplicates from a list and converts it to a set.
5. **Mutability and Immutability**
   (a) Develop a function that modifies a list by squaring each element.
   (b) Write a Python script that attempts to modify an element in a tuple and handles the error gracefully.
6. **Type Casting**
   (a) Create a program that prompts the user to enter a number as a string, converts it to an integer, squares the result, and prints the squared value.
   (b) Implement a function that takes a list of numbers as strings and returns their sum as an integer.

LAB ACTIVITY 3

CONTROL STATEMENTS IN PYTHON-I

1. **Leap Year Checker:** Write a Python program that takes a year as input and determines whether it is a leap year. Consider the rules for leap years (divisible by 4, but not divisible by 100 unless divisible by 400).

2. **Temperature Converter:** Implement a temperature converter program in Python. Prompt the user to enter a temperature in Celsius and convert it to Fahrenheit. Provide feedback based on whether the temperature is considered hot, moderate, or cold.

3. **Number Classifier:** Create a Python program that classifies a given integer as positive, negative, or zero. Additionally, check if the number is even or odd and provide appropriate feedback.

4. **Vowel or Consonant:** Write a Python program that takes a single character as input and determines whether it is a vowel or a consonant. Handle both uppercase and lowercase characters.

5. **Simple Calculator:** Develop a simple calculator program in Python that takes two numbers and an operator (+, -, *, /) as input and performs the corresponding operation. Handle division by zero gracefully.

6. **Discount Calculator:** Create a Python program that calculates the final price after applying a discount. Prompt the user to enter the original price and the discount percentage. Display the discounted price.

7. **Guess the Number:** Implement a number guessing game in Python. Generate a random number between 1 and 100 and ask the user to guess the number. Provide hints (higher/lower) until the correct number is guessed.

8. **Triangle Classifier:** Write a Python program that takes three sides of a triangle as input and classifies it as equilateral, isosceles, or scalene. Handle cases where the input sides cannot form a valid triangle.

9. **BMI Calculator:** Develop a Python program that calculates the Body Mass Index (BMI) based on the user's weight (in kilograms) and height (in meters). Provide feedback on whether the user is underweight, normal weight, overweight, or obese.

10. **Time Converter:** Create a time converter program in Python that converts seconds into hours, minutes, and seconds. Prompt the user to enter the total seconds and display the converted time.

LAB ACTIVITY 4

CONTROL STATEMENTS IN PYTHON-II

1. **Factorial Calculator:**
    - Write a Python program that calculates the factorial of a given positive integer using a loop.
    - Prompt the user to input a number, calculate its factorial, and print the result.
2. **Palindrome Checker:**
    - Implement a Python program that checks whether a given string is a palindrome (reads the same backward as forward).
    - Use a loop to iterate through the characters and determine if the string is a palindrome.
3. **Multiplication Table Generator:**
    - Create a Python program that generates the multiplication table for a given integer.
    - Use a loop to print the table from 1 to 10 for the specified number.
4. **Fibonacci Sequence Generator:**
    - Write a Python program that generates the Fibonacci sequence up to a specified limit.
    - Use a loop to calculate and print the Fibonacci numbers until reaching the given limit.
5. **Prime Number Checker:**
    - Develop a Python program that checks whether a given number is a prime number.
    - Use a loop to iterate through possible divisors and determine the primality of the input.
6. **Pattern Printing - Pyramid:**
    - Implement a Python program that prints a pyramid pattern using nested loops.
    - Prompt the user to enter the height of the pyramid and print the pattern accordingly.
7. **Rock, Paper, Scissors Game:**
    - Write a Python program that simulates a rock, paper, scissors game against the computer.
    - Use a loop to allow the user to play multiple rounds until they choose to exit.
8. **Power Calculator:**
    - Develop a Python program that calculates the result of raising a number to a specified power using a loop.
    - Prompt the user to input the base and exponent, and print the result.

LAB ACTIVITY 5

APPLICATION: MATRICES

1. **Transpose Function:**
    - Implement a Python function that takes a matrix as input and returns its transpose.
    - Test the function with different matrices.
2. **Matrix Determinant Calculator:**
    - Write a Python program that calculates the determinant of a 2x2 or 3x3 matrix.
    - Implement a function to handle matrix input and display the determinant.
3. **Matrix Inversion:**
    - Create a Python program that finds the inverse of a 2x2 matrix.
    - Implement a function that takes a matrix as input and returns its inverse.
4. **Matrix Trace Calculator:**
    - Develop a Python program that calculates the trace of a square matrix.
    - The trace of a matrix is the sum of its diagonal elements.
5. **Sparse Matrix Representation:**
    - Implement a Python program to represent a sparse matrix.
    - Perform addition or multiplication operations on sparse matrices.
6. **Eigenvalue and Eigenvector Solver:**
    - Write a Python program that finds the eigenvalues and eigenvectors of a square matrix.
    - Implement functions to calculate eigenvalues and eigenvectors.
7. **Matrix Rotation:**
    - Create a Python program that rotates a given matrix 90 degrees clockwise.
    - Allow the user to input the matrix dimensions and elements.
8. **Matrix Rank Calculator:**
    - Write a Python program that calculates the rank of a matrix.
    - Implement a function to determine the rank of a given matrix.

LAB ACTIVITY 6

APPLICATION: DETERMINANTS, SYSTEM OF LINEAR EQUATIONS

1. **Determinant Calculation (2x2) with NumPy**
   Write a Python function that takes a 2×2 matrix as a NumPy array and calculates its determinant using the NumPy function `numpy.linalg.det()`.

2. **Determinant Calculation (3x3) with NumPy**
   Implement a Python function that calculates the determinant of a given 3×3 matrix (provided as a NumPy array) using the NumPy function `numpy.linalg.det()`.

3. **Matrix Invertibility Check with NumPy**
   Create a Python function that determines if a square matrix (given as a NumPy array) is invertible based on its determinant using the NumPy functions. The function should return `True` if invertible and `False` otherwise.

4. **Matrix Inverse Calculation with NumPy**
   Write a Python function to find the inverse of a given invertible matrix (provided as a NumPy array) using the NumPy function `numpy.linalg.inv()`. If the matrix is not invertible, handle the exception and print an appropriate message.

5. **Solving Linear Equations (2x2 System) with NumPy**
   Implement a Python function that solves a system of two linear equations using determinants. The input should be the coefficient matrix and constants vector, both provided as NumPy arrays.

6. **Solving Linear Equations (3x3 System) with NumPy**
   Extend the previous function to handle a system of three linear equations. Implement the necessary calculations for a 3×3 system using NumPy functions.

7. **Application in Geometry with NumPy**
   Create a Python function that takes a 2×2 transformation matrix as a NumPy array and uses the determinant with NumPy to determine whether the transformation involves a reflection or a rotation.

8. **Laplace Expansion for Larger Matrices with NumPy**
   Write a Python function that applies Laplace Expansion to calculate the determinant of a given 4×4 matrix (provided as a NumPy array). Implement the recursive process for cofactor expansion using NumPy functions.

9. **Random Matrix System Solver with NumPy**
   Generate a random 3×3 matrix and a constants vector using NumPy. Write a Python function to solve the corresponding system of linear equations using determinants and NumPy functions.

LAB ACTIVITY 7

APPLICATION: SYSTEM OF EQUATIONS

1. **Determinant Calculation:** Write a Python function to calculate the determinant of a given square matrix. Test your function with a 3x3 matrix.
2. **System Classification:** Create a Python function to determine if a given coefficient matrix represents a system of equations with a unique solution. Use this function to analyze a sample matrix.
3. **NumPy Solution:** Implement a Python program using NumPy to solve a system of linear equations. Represent the system with a coefficient matrix and a constants vector.
4. **Symbolic Solution:** Extend the previous program to use SymPy for symbolic computation. Allow users to input variables and coefficients symbolically.
5. **Multiple Solutions:** Write a Python function to handle systems of equations with multiple solutions. Test it with a matrix that has infinitely many solutions.
6. **Error Handling:** Modify the NumPy solution program to include appropriate error handling, e.g., for the case where the determinant is zero.
7. **Real-world Application:** Devise a scenario where solving a system of equations is necessary in a real-world context. Write a Python program that models and solves this scenario using NumPy.
8. **Comparative Analysis:** Create a Python program that solves a system of equations using both NumPy and SymPy. Compare the results and discuss the advantages and limitations of each approach.
9. **Extended Systems:** Extend the second sample program to handle larger systems of equations (e.g., 4x4 or 5x5). Test it with a relevant example.
10. **Interactive Solver:** Develop an interactive Python script that takes user input for a system of equations. Dynamically determine and display the solution using NumPy.

LAB ACTIVITY 8

APPLICATION: EIGENVALUES, EIGENVECTORS

1. **Eigenvalue and Eigenvector Computation with NumPy**
   Write a Python function that takes a square matrix as input, uses NumPy to compute its eigenvalues and eigenvectors, and returns the results.

2. **Matrix Classification**
   Create a Python function that takes a square matrix as input and determines if it is positive definite, positive semi-definite, negative definite, or negative semi-definite. Use NumPy to compute eigenvalues for the classification.

3. **Cholesky Decomposition (if applicable)**
   If the matrix from the previous exercise is positive definite or negative definite, extend the function to perform Cholesky decomposition using NumPy and return the lower triangular matrix.

4. **Power Iteration Method**
   Write a Python function to find the dominant eigenvalue and corresponding eigenvector of a matrix using the Power Iteration method. Ensure convergence and handle potential errors.

5. **Iterative Eigensolver**
   Write a Python function to find several eigenvalues and corresponding eigenvectors using an iterative method (e.g., the QR algorithm). Allow the user to specify the number of eigenvalues to find.

LAB ACTIVITY 9

APPLICATION: ROOTS OF EQUATIONS

1. Consider the quadratic equation:

$$f(x) = 3x^2 - 12x + 12$$

   (a) Find the numerical roots of the equation using NumPy's `np.roots` function.

   (b) Determine if the equation has real roots, complex roots, or repeated roots based on the discriminant.

   (c) Find the symbolic roots of the equation using SymPy's `sp.solve` function.

   (d) Verify the numerical roots by substituting each root back into the original equation and checking if the result is close to zero.

2. Given the trigonometric equation:

$$g(x) = \cos(x) - \sin(x)$$

   (a) Find the numerical roots of the equation using NumPy's `np.roots` function.

   (b) Determine the periodicity or symmetry properties of the trigonometric equation.

   (c) Find the symbolic roots of the equation using SymPy's `sp.solve` function.

   (d) Verify the numerical roots by substituting each root back into the original equation and checking if the result is close to zero.

3. Consider the cubic equation:

$$h(x) = x^3 - 6x^2 + 11x - 6$$

   (a) Find the numerical roots of the equation using NumPy's `np.roots` function.

   (b) Determine the multiplicity of each root.

   (c) Find the symbolic roots of the equation using SymPy's `sp.solve` function.

   (d) Verify the numerical roots by substituting each root back into the original equation and checking if the result is close to zero.

4. Given the equation:

$$k(x) = e^{2x} - 3e^x + 2$$

(a) Find the numerical roots of the equation using NumPy's `np.roots` function.

(b) Determine the behavior of the function for large positive and negative values of x.

(c) Find the symbolic roots of the equation using SymPy's `sp.solve` function.

(d) Verify the numerical roots by substituting each root back into the original equation and checking if the result is close to zero.

5. Consider the following system of nonlinear equations:

$$\begin{aligned} 2x^2 + y &= 4 \\ x + y^2 &= 3 \end{aligned}$$

(a) Formulate the system as a vectorized function and find the numerical solution using NumPy.

(b) Determine if the system has a unique solution.

(c) Find the symbolic solution using SymPy.

(d) Verify the numerical solution by substituting it back into the original system of equations.

6. Given the polynomial equation:

$$p(x) = 5x^4 - 6x^3 - 4x^2 + 2x - 3$$

(a) Find the numerical roots of the equation using NumPy's `np.roots` function.

(b) Identify any complex roots and express them in the form $a + bi$.

(c) Find the symbolic roots of the equation using SymPy's `sp.solve` function.

(d) Verify the numerical roots by substituting each root back into the original equation and checking if the result is close to zero.

7. Consider the trigonometric equation:

$$q(x) = \sin(2x) + \cos(x)$$

(a) Find the numerical roots of the equation using NumPy's `np.roots` function.

(b) Determine the periodicity or symmetry properties of the trigonometric equation.

(c) Find the symbolic roots of the equation using SymPy's `sp.solve` function.

(d) Verify the numerical roots by substituting each root back into the original equation and checking if the result is close to zero.

8. Given the equation:

$$r(x) = \sqrt{x} - \log(x)$$

(a) Find the numerical roots of the equation using NumPy's `np.roots` function.

(b) Determine the domain of the function.

(c) Find the symbolic roots of the equation using SymPy's `sp.solve` function.

(d) Verify the numerical roots by substituting each root back into the original equation and checking if the result is close to zero.

9. Consider the system of equations:

$$3x - y = 5$$
$$x^2 + y^2 = 10$$

   (a) Formulate the system as a vectorized function and find the numerical solution using NumPy.

   (b) Determine if the system has a unique solution.

   (c) Find the symbolic solution using SymPy.

   (d) Verify the numerical solution by substituting it back into the original system of equations.

10. Given the equation:

$$s(x) = \tan(x) - 2x$$

   (a) Find the numerical roots of the equation using NumPy's `np.roots` function.

   (b) Determine the periodicity or symmetry properties of the trigonometric equation.

   (c) Find the symbolic roots of the equation using SymPy's `sp.solve` function.

   (d) Verify the numerical roots by substituting each root back into the original equation and checking if the result is close to zero.

LAB ACTIVITY 10

APPLICATION: NUMERICAL INTEGRATION

1. Given the function $g(x) = \sin(x)$ over the interval $[0, \pi]$.
   (a) Use SciPy's `integrate.quad` for numerical integration.
   (b) Try different methods in `integrate.quad`.
   (c) Compare the results with the analytical solution.
   (d) Analyze the accuracy for oscillatory functions.
2. Consider the function $h(x) = \frac{1}{1+x^2}$ over the interval $[0, 5]$.
   (a) Apply SciPy's `integrate.quad` for numerical integration.
   (b) Experiment with various methods provided by `integrate.quad`.
   (c) Compare the results with known analytical solutions.
   (d) Assess the impact of function behavior on integration accuracy.
3. Given the function $k(x) = \sqrt{1-x^2}$ over the interval $[-1, 1]$.
   (a) Utilize SciPy's `integrate.quad` for numerical integration.
   (b) Explore different methods available in `integrate.quad`.
   (c) Compare the results with the analytical solution.
   (d) Investigate the impact of changing the interval.
4. Consider the function $p(x) = x^3 - 2x^2 + x$ over the interval $[0, 2]$.
   (a) Perform numerical integration using SciPy's `integrate.quad`.
   (b) Experiment with various methods provided by `integrate.quad`.
   (c) Compare the results with the analytical solution if available.
   (d) Analyze the behavior of the integrand on accuracy.
5. Given the function $q(x) = \ln(x)$ over the interval $[1, 2]$.
   (a) Apply SciPy's `integrate.quad` for numerical integration.
   (b) Experiment with different methods in `integrate.quad`.
   (c) Compare the results with the analytical solution.
   (d) Assess the impact of function behavior on integration accuracy.
6. Consider the function $r(x) = \cos(x)$ over the interval $[0, \frac{\pi}{2}]$.

(a) Utilize SciPy's `integrate.quad` for numerical integration.

(b) Experiment with various methods in `integrate.quad`.

(c) Compare the results with the analytical solution.

(d) Investigate the impact of changing the interval.

7. Given the function $s(x) = x^2 + 2x$ over the interval $[-1, 1]$.

(a) Perform numerical integration using SciPy's `integrate.quad`.

(b) Experiment with different methods provided by `integrate.quad`.

(c) Compare the results with the analytical solution if available.

(d) Analyze the impact of function behavior on accuracy.

8. Consider the function $t(x) = e^x$ over the interval $[0, 1]$.

(a) Apply SciPy's `integrate.quad` for numerical integration.

(b) Experiment with various methods in `integrate.quad`.

(c) Compare the results with the analytical solution.

(d) Assess the impact of changing the interval.

9. Given the function $u(x) = \frac{\sin(x)}{x}$ over the interval $[0, \pi]$.

(a) Utilize SciPy's `integrate.quad` for numerical integration.

(b) Experiment with different methods in `integrate.quad`.

(c) Compare the results with the analytical solution.

(d) Investigate the behavior of the integrand on accuracy.

LAB ACTIVITY 11

LIST COMPREHENSIONS

1. **Squares List:** Create a list of the squares of numbers from 1 to 10 using a list comprehension.
2. **Filter Even Numbers:** Generate a list of even numbers from 1 to 20 using a list comprehension.
3. **Nested List Comprehension:** Create a 2D matrix (list of lists) with elements as tuples (x, y) for all x and y from 1 to 3 using nested list comprehensions.
4. **Conditional List Comprehension:** Generate a list of cubes of numbers from 1 to 10 but only include cubes that are divisible by 4.
5. **String Transformation:** Given a list of strings, create a new list containing the lengths of strings that have more than three characters.
6. **List Transformation:** Given a list of integers, create a new list with each element doubled using a list comprehension.
7. **List of Tuples:** Create a list of tuples where each tuple consists of an integer and its square, for integers from 1 to 5.
8. **Prime Numbers:** Generate a list of prime numbers between 1 and 50 using a list comprehension.
9. **Character Frequencies:** Given a string, create a dictionary containing the frequency of each character using a list comprehension.
10. **Matrix Transposition:** Given a 2D matrix represented as a list of lists, transpose the matrix using a list comprehension.

LAB ACTIVITY 12

GRAPH PLOTTING

1. **Linear Function Plotter:**
   Write a Python program that takes the coefficients of a linear function (e.g., $y = mx + b$) as input and plots the corresponding graph using Matplotlib.

2. **Quadratic Function Animation:**
   Create an animated plot that shows the changing graph of a quadratic function (e.g., $y = ax^2 + bx + c$) as the coefficients vary. Allow users to interactively adjust the coefficients.

3. **Function Composition:**
   Implement a program that combines a linear and a quadratic function into a single plot. Allow users to choose the coefficients and visualize the resulting graph.

4. **Customized Plot Styling:**
   Develop a Python script that demonstrates advanced customization of Matplotlib plots. Include features such as different line styles, colors, and markers. Apply these to the linear, quadratic, and exponential functions.

5. **Multiple Function Plots:**
   Create a program that plots multiple functions on the same graph. Allow users to input multiple functions and display them with distinct colors and labels.

6. **Interactive Exponential Function:**
   Build an interactive plot for the exponential function where users can input a range for x values, and the plot updates dynamically. Include axis labels, a legend, and a title.

7. **Polynomial Function Plotter:**
   Develop a versatile Python tool that can plot polynomials of any degree. Allow users to input coefficients, specify the degree, and visualize the resulting polynomial graph.

8. **Logarithmic Scale Plot:**
   Modify the exponential function plot to be displayed on a logarithmic scale. Demonstrate how to set up logarithmic axes using Matplotlib.

9. **Function Derivative Visualization:**
   Extend the linear function plot program to also display the derivative of the function. Use numerical differentiation to calculate the derivative.

10. **Error Handling in Plotting:**
    Implement a robust Python script that handles potential errors in user input for function coefficients. Provide informative error messages and guide the user to correct any mistakes.

LAB ACTIVITY 13

DATA VISUALIZATION

1. Line Plot with Multiple Lines Modify the line plot example to include two or more lines with different colors and styles.
2. Customizing Scatter Plot Create a scatter plot with random data and customize it by adding a title, labels, and changing the marker style.
3. Grouped Bar Chart Extend the bar chart example to create a grouped bar chart with multiple categories and compare values.
4. Box Plot with Seaborn Use Seaborn to create a box plot with a dataset of your choice, and customize it with appropriate labels and titles.
5. Kernel Density Estimation Plot Experiment with different datasets and create a distribution plot using kernel density estimation with Seaborn.
6. Enhanced Pair Plot Improve the pair plot example by customizing the appearance, adding labels, and highlighting different classes.

LAB ACTIVITY 14

DICTIONARY AND SORTING, MINIMUM AND MAXIMUM

1. **Accessing Dictionary Values:** Create a dictionary representing a product with attributes like name, price, and quantity. Allow the user to input the product name and display its details.

2. **Sorting by Keys:** Given a dictionary with random key-value pairs, write a function to sort the dictionary based on keys and print the sorted result.

3. **Sorting by Values:** Write a Python program to sort a dictionary based on values in descending order. Display the sorted dictionary.

4. **Minimum and Maximum Keys:** Create a dictionary with keys as years (e.g., 2020, 2021, 2022) and values as some data. Find and display the minimum and maximum years.

5. **Minimum and Maximum Values:** Develop a program that defines a dictionary representing temperatures in Celsius for different days. Find and print the day with the minimum and maximum temperature.

6. **Nested Dictionaries:** Extend the student dictionary example to include nested dictionaries for each student, representing their courses and corresponding grades.

7. **Filtering Dictionary:** Given a dictionary of product prices, write a function to filter out products with prices below a certain threshold. Display the filtered dictionary.

8. **Updating Dictionary:** Implement a program where the user can update the details of a student in the dictionary (e.g., change the grade or age).

9. **Dictionary Operations:** Write a Python program that performs various operations on dictionaries, such as merging two dictionaries, removing a key-value pair, and clearing the entire dictionary.