Channabasaveshwara Institute of Technology
(Affiliated to VTU, Belgaum & Approved by AICTE, New Delhi)
(NAAC Accredited & ISO 9001:2015 Certified Institution)
NH 206 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka

LABORATORY MANUAL
PYTHON PROGRAMMING

Course Code: 1BPLC105B / 1BPLC205B
Semester: I / II Semester B.E. / B.Tech.
Credits: 4 (Theory: 3 + Lab: 1)
CIE Marks: 50 | SEE Marks: 50 | Total: 100
Lab Hours/Week: 02 Hours (Practical)
University: Vishwesvaraya Technological University (VTU), Belagavi

Department of Computer Science & Engineering

Prepared By:
Dr. Anil Kumar G
Professor and Dean Academics

Mrs. Vindhya G B
Assistant Professor

Reviewed and Approved By:
Dr. Shanthala C
Professor and HEAD

Course Overview

Course Code: 1BPLC105B (Semester I) / 1BPLC205B (Semester II)

Teaching Hours: L:T:P:S = 3:0:2:0 | Total Theory Hours: 40 | Total Lab Hours: 24

Credits: 4 | CIE: 50 Marks | SEE: 50 Marks | Total: 100 Marks

Course Outcomes (COs)

1. CO1: Develop scripts using primitive language constructs of Python.
2. CO2: Identify the methods to manipulate primitive Python data structures.
3. CO3: Make use of Python standard libraries for programming.
4. CO4: Build scripts for performing file operations.
5. CO5: Illustrate the concepts of Object-Oriented Programming as used in Python.

Assessment Structure

Component	Marks	Minimum to Pass
CIE Theory (IA Tests)	25	10 (40%)
CIE Practical (Rubrics + Lab IA)	25	10 (40%)
SEE (Semester End Exam)	50	18 (35%)
Total	100	40 (40%)

Index of Experiments

General Lab Instructions

- Write each program in the lab record before execution.
- Ensure Python 3.x is installed. Use IDLE, VS Code, or Jupyter Notebook.
- Observe proper indentation — Python is indentation-sensitive.
- Save programs with meaningful names, e.g., exp1a_arithmetic.py.
- Test your program with multiple inputs including edge cases.
- Record algorithm, code, and output in the lab observation book.
- Do not copy programs — writing code builds understanding.
- Raise any issues with hardware/software to the lab instructor promptly.

Experiment 1a: Basic Arithmetic Operations Based on User Choice

Aim	Develop a Python program to read 2 numbers from the keyboard and perform the basic arithmetic operations based on the choice. (1-Add, 2-Subtract, 3-Multiply, 4-Divide).
Objective	To practice reading user input, conditional branching with if-elif-else, and basic arithmetic operators.

Program Code

```
# Experiment 1a: Arithmetic Operations based on Choice

print('========== Arithmetic Calculator ==========')
num1 = float(input('Enter first number : '))
num2 = float(input('Enter second number: '))
print('\n Select Operation:')
print('  1 - Add')
print('  2 - Subtract')
print('  3 - Multiply')
print('  4 - Divide')
choice = int(input('\n Enter your choice (1-4): '))

if choice == 1:
    result = num1 + num2
    op = '+'
elif choice == 2:
    result = num1 - num2
    op = '-'
elif choice == 3:
    result = num1 * num2
    op = '*'
elif choice == 4:
    if num2 == 0:
        print('Error: Division by zero is not allowed!')
    else:
        result = num1 / num2
        op = '/'
        print(f'Result: {num1} {op} {num2} = {result}')
else:
    print('Invalid choice! Please enter a number between 1 and 4.')
```

```
if choice in [1, 2, 3]:
    print(f'Result: {num1} {op} {num2} = {result}')
```

Sample Output

```
========== Arithmetic Calculator ==========
Enter first number : 25
Enter second number: 4

Select Operation:
  1 - Add
  2 - Subtract
  3 - Multiply
  4 - Divide

Enter your choice (1-4): 4
Result: 25.0 / 4.0 = 6.25
```

Experiment 1b: Senior Citizen Check

*

Aim	Develop a program to read the name and year of birth of a person. Display whether the person is a senior citizen or not.
Objective	To understand variable input, arithmetic with the current year, and conditional decision making.

Program Code

```
# Experiment 1b: Senior Citizen Check

CURRENT_YEAR = 2025
SENIOR_AGE   = 60

name = input('Enter your name         : ')
yob  = int(input('Enter your year of birth: '))

age = CURRENT_YEAR - yob

print(f'\nName : {name}')
print(f'Age  : {age} years')

if age >= SENIOR_AGE:
    print(f'{name} IS a Senior Citizen (age >= {SENIOR_AGE}).')
else:
    remaining = SENIOR_AGE - age
    print(f'{name} is NOT a Senior Citizen.')
    print(f'{name} will become a Senior Citizen in {remaining} year(s).')
```

Sample Output

```
Enter your name         : Ramesh
Enter your year of birth: 1958

Name : Ramesh
Age  : 67 years
Ramesh IS a Senior Citizen (age >= 60).
```

Experiment 2a: Fibonacci Sequence of Length N

Aim	Develop a program to generate Fibonacci sequence of length (N). Read N from the console.
Objective	To practice loop constructs (while / for) and understand the Fibonacci pattern.

Program Code

```
# Experiment 2a: Fibonacci Sequence of Length N

N = int(input('Enter the length of Fibonacci sequence (N): '))

if N <= 0:
    print('Please enter a positive integer.')
else:
    a, b = 0, 1
    fib_series = []
    for _ in range(N):
        fib_series.append(a)
        a, b = b, a + b

    print(f'\nFibonacci sequence of length {N}:')
    print(' ->'.join(map(str, fib_series)))
```

Sample Output

```
Enter the length of Fibonacci sequence (N): 10

Fibonacci sequence of length 10:
0 -> 1 -> 1 -> 2 -> 3 -> 5 -> 8 -> 13 -> 21 -> 34
```

Experiment 2b: List Operations

Aim	Write a Python program to create a list and perform the following operations: Inserting an element, Removing an element, Appending an element, Displaying the length of the list, Popping an element, Clearing the list.
Objective	To learn the built-in list methods: insert(), remove(), append(), len(), pop(), and clear().

Program Code

```
# Experiment 2b: List Operations

my_list = [10, 20, 30, 40, 50]
print(f'Initial List: {my_list}')

# Insert element at position 2
my_list.insert(2, 25)
print(f'After insert(2, 25): {my_list}')

# Remove element with value 40
my_list.remove(40)
print(f'After remove(40)     : {my_list}')

# Append element at the end
my_list.append(60)
print(f'After append(60)     : {my_list}')

# Display length
print(f'Length of list       : {len(my_list)}')

# Pop last element
popped = my_list.pop()
print(f'Popped element       : {popped}')
print(f'List after pop()     : {my_list}')

# Clear the list
my_list.clear()
print(f'List after clear()   : {my_list}')
```

Sample Output

```
Initial List        : [10, 20, 30, 40, 50]
After insert(2, 25)  : [10, 20, 25, 30, 40, 50]
After remove(40)     : [10, 20, 25, 30, 50]
After append(60)     : [10, 20, 25, 30, 50, 60]
Length of list       : 6
Popped element       : 60
List after pop()     : [10, 20, 25, 30, 50]
List after clear()   : []
```

Experiment 3a: Mean, Variance and Standard Deviation

Aim	Read N numbers from the console and create a list. Develop a program to print mean, variance and standard deviation with suitable messages.
Objective	To apply list operations, mathematical formulas, and understand basic statistical computations in Python.

Program Code

```
# Experiment 3a: Mean, Variance and Standard Deviation

import math

N = int(input('Enter number of values (N): '))
numbers = []
for i in range(N):
    val = float(input(f'  Enter value {i+1}: '))
    numbers.append(val)

# Mean
mean = sum(numbers) / N

# Variance
variance = sum((x - mean) ** 2 for x in numbers) / N

# Standard Deviation
std_dev = math.sqrt(variance)

print(f'\nList of Numbers    : {numbers}')
print(f'Mean               : {mean:.4f}')
print(f'Variance           : {variance:.4f}')
print(f'Standard Deviation : {std_dev:.4f}')
```

Sample Output

```
Enter number of values (N): 5
  Enter value 1: 10
  Enter value 2: 20
  Enter value 3: 30
  Enter value 4: 40
  Enter value 5: 50

List of Numbers    : [10.0, 20.0, 30.0, 40.0, 50.0]
Mean               : 30.0000
Variance           : 200.0000
Standard Deviation : 14.1421
```

Experiment 3b: Frequency of Each Digit in a Multi-Digit Number

Aim	Read a multi-digit number (as chars) from the console. Develop a program to print the frequency of each digit with a suitable message.
Objective	To practice string traversal, dictionary usage, and character frequency counting.

Program Code

```
# Experiment 3b: Frequency of Each Digit

number = input('Enter a multi-digit number: ')

# Validate: only digits allowed
if not number.isdigit():
    print('Invalid input! Please enter digits only.')
else:
    freq = {}
    for ch in number:
        freq[ch] = freq.get(ch, 0) + 1

    print(f'\nNumber entered : {number}')
    print(f'Total digits   : {len(number)}')
    print('\nDigit  |  Frequency')
    print('-' * 20)
    for digit in sorted(freq.keys()):
        print(f'  {digit}    |    {freq[digit]}')
```

Sample Output

```
Enter a multi-digit number: 1122334455

Number entered : 1122334455
Total digits   : 10

Digit  |  Frequency
--------------------
  1    |    2
  2    |    2
  3    |    2
  4    |    2
  5    |    2
```

Experiment 4: 10 Most Frequently Appearing Words in a Text File

Aim	Develop a program to print 10 most frequently appearing words in a text file. [Hint: Use a dictionary with distinct words and their frequency of occurrences. Sort the dictionary in the reverse order of frequency and display the dictionary slice of the first 10 items.]
Objective	To practice file I/O, dictionary construction, string methods, and sorting with lambda key.

▸ Program Code

```
# Experiment 4: 10 Most Frequently Appearing Words in a Text File
# NOTE: Create a sample text file 'sample.txt' before running.

filename = input('Enter text file name (e.g., sample.txt): ')

try:
    with open(filename, 'r') as f:
        content = f.read().lower()

    # Remove punctuation and split into words
    import string
    for ch in string.punctuation:
        content = content.replace(ch, '')
    words = content.split()

    # Build frequency dictionary
    freq = {}
    for word in words:
        freq[word] = freq.get(word, 0) + 1

    # Sort by frequency (descending) and take top 10
    sorted_freq = sorted(freq.items(), key=lambda x: x[1], reverse=True)
    top10 = sorted_freq[:10]

    print(f'\nTotal unique words : {len(freq)}')
    print(f'Total words       : {len(words)}')
    print('\nTop 10 Most Frequent Words:')
    print(f'{"Rank":<6}{"Word":<20}{"Frequency"}')
    print('-' * 36)
    for rank, (word, count) in enumerate(top10, 1):
        print(f'{rank:<6}{word:<20}{count}')
```

```
except FileNotFoundError:
    print(f'Error: File "{filename}" not found!')
```

Sample Output

```
Enter text file name (e.g., sample.txt): sample.txt

Total unique words : 48
Total words        : 120

Top 10 Most Frequent Words:
Rank  Word                Frequency
-----------------------------------
1     the                 12
2     is                  8
3     python              7
4     and                 6
5     a                   5
6     of                  5
7     to                  4
8     in                  4
9     for                 3
10    programming         3
```

Experiment 5: Sort 6 Subject Marks using Bubble Sort (Highest to Lowest)

Aim	Develop a program to read 6 subject marks from the keyboard for a student. Generate a report that displays the marks from the highest to the lowest score attained by the student. [Read the marks into a 1-Dimensional array and sort using the Bubble Sort technique].
Objective	To implement Bubble Sort manually on a 1D list and generate a formatted academic report.

▸ Program Code

```
# Experiment 5: Bubble Sort on Student Marks (Highest to Lowest)

SUBJECTS = ['Mathematics', 'Physics', 'Chemistry',
'English', 'Computer Science', 'Electronics']

print('========== Student Mark Entry ==========')
student_name = input('Enter student name: ')
marks = []
for sub in SUBJECTS:
    m = int(input(f'Enter marks for {sub} (0-100): '))
    marks.append(m)

# Bubble Sort - Descending order
n = len(marks)
sorted_marks = marks[:]      # copy
for i in range(n - 1):
    for j in range(0, n - i - 1):
        if sorted_marks[j] < sorted_marks[j + 1]:
            sorted_marks[j], sorted_marks[j+1] = sorted_marks[j+1], sorted_marks[j]

# Report
print(f'\n======= Report Card: {student_name} =======')
print(f'{"Rank":<6}{"Subject":<20}{"Marks"}')
print('-' * 36)
for rank, mark in enumerate(sorted_marks, 1):
    sub_idx = marks.index(mark)
    print(f'{rank:<6}{SUBJECTS[sub_idx]:<20}{mark}')
    marks[sub_idx] = -1   # Mark as used
print('-' * 36)
print(f'Total Marks : {sum(sorted_marks)} / 600')
print(f'Percentage  : {sum(sorted_marks)/6:.2f}%')
```

Sample Output

```
========== Student Mark Entry ==========
Enter student name: Aditya
Enter marks for Mathematics (0-100): 88
Enter marks for Physics (0-100): 76
Enter marks for Chemistry (0-100): 92
Enter marks for English (0-100): 65
Enter marks for Computer Science (0-100): 95
Enter marks for Electronics (0-100): 80

======= Report Card: Aditya =======
Rank  Subject             Marks
-----------------------------------
1     Computer Science    95
2     Chemistry           92
3     Mathematics         88
4     Electronics         80
5     Physics             76
6     English             65
-----------------------------------
Total Marks : 496 / 600
Percentage  : 82.67%
```

Experiment 6: Sort Contents of a Text File and Write to Another File

Aim	Develop a program to sort the contents of a text file and write the sorted contents into a separate text file. [Hint: Use string methods strip(), len(), list methods sort(), append(), and file methods open(), readlines(), and write()].
Objective	To practice file reading, list sorting, and writing sorted output to a new file using standard file methods.

Program Code

```
# Experiment 6: Sort Text File Contents into Another File

input_file  = input('Enter input  file name: ')
output_file = input('Enter output file name: ')

try:
    # Read lines from input file
    with open(input_file, 'r') as fin:
        lines = fin.readlines()

    if len(lines) == 0:
        print('The file is empty!')
    else:
        # Strip whitespace/newlines and filter blank lines
        words = []
        for line in lines:
            for word in line.strip().split():
                words.append(word.strip())

        words.sort()    # Sort alphabetically

        # Write sorted content to output file
        with open(output_file, 'w') as fout:
            fout.write('Sorted Contents:\n')
            fout.write('=' * 30 + '\n')
            for word in words:
                fout.write(word + '\n')

        print(f'\nTotal words sorted : {len(words)}')
        print(f'Sorted contents written to "{output_file}" successfully.')
        print(f'\nFirst 10 sorted words:')
```

```
        for w in words[:10]:
            print(f'  {w}')

except FileNotFoundError:
    print(f'Error: File "{input_file}" not found!')
```

Sample Output

```
Enter input  file name: input.txt
Enter output file name: sorted_output.txt

Total words sorted : 20
Sorted contents written to "sorted_output.txt" successfully.

First 10 sorted words:
  Banana
  Cherry
  Grapes
  Kiwi
  Mango
  Orange
  Papaya
  Peach
  Pear
  Pineapple
```

Experiment 7: DivExp Function with Assertion and Exception Handling

Aim	Develop a function named DivExp which takes TWO parameters a, b, and returns a value c (c=a/b). Write a suitable assertion for a>0 in the function DivExp and raise an exception for when b=0. Develop a suitable program that reads two console values and calls the function DivExp.
Objective	To understand assertions, raising custom exceptions, try-except-finally blocks, and error-safe function design.

▸ Program Code

```
# Experiment 7: DivExp with Assertion and Exception Handling

def DivExp(a, b):
"""Divides a by b. Asserts a > 0, raises exception if b == 0."""
    assert a > 0, f'Assertion failed: a must be > 0, but got a = {a}'
    if b == 0:
        raise ZeroDivisionError('Error: Denominator b cannot be zero!')
    c = a / b
    return c

try:
    a = float(input('Enter value of a (must be > 0): '))
    b = float(input('Enter value of b (divisor)   : '))
    result = DivExp(a, b)
    print(f'\nDivExp({a}, {b}) = {result:.4f}')

except AssertionError as ae:
    print(f'\nAssertionError: {ae}')
except ZeroDivisionError as ze:
    print(f'\nZeroDivisionError: {ze}')
except ValueError:
    print('\nValueError: Please enter valid numeric values!')
finally:
    print('Program execution complete.')
```

Sample Output

```
--- Run 1: Normal case ---
Enter value of a (must be > 0): 10
Enter value of b (divisor)   : 4

DivExp(10.0, 4.0) = 2.5000
Program execution complete.

--- Run 2: b = 0 ---
Enter value of a (must be > 0): 10
Enter value of b (divisor)   : 0

ZeroDivisionError: Error: Denominator b cannot be zero!
Program execution complete.

--- Run 3: a <= 0 ---
Enter value of a (must be > 0): -5
Enter value of b (divisor)   : 2

AssertionError: Assertion failed: a must be > 0, but got a = -5.0
Program execution complete.
```

Experiment 8: Addition of N Complex Numbers using a Class

Aim	Define a function that takes TWO objects representing complex numbers and returns a new complex number with the sum of two complex numbers. Define a suitable class 'Complex' to represent the complex number. Develop a program to read N (N >= 2) complex numbers and to compute the addition of N complex numbers.
Objective	To understand class definition, object instantiation, operator overloading (__add__, __str__), and working with instances.

▸ Program Code

```
# Experiment 8: Complex Number Addition using a Class

class Complex:
    def __init__(self, real, imag):
        self.real = real
        self.imag = imag

    def __add__(self, other):
        return Complex(self.real + other.real, self.imag + other.imag)

    def __str__(self):
        sign = '+' if self.imag >= 0 else '-'
        return f'{self.real} {sign} {abs(self.imag)}i'

def add_complex(c1, c2):
"""Returns sum of two Complex objects."""
    return c1 + c2

N = int(input('Enter number of complex numbers (N >= 2): '))
if N < 2:
    print('N must be >= 2')
else:
    complex_list = []
    for i in range(N):
        r = float(input(f'  Enter real part of complex {i+1}     : '))
        im = float(input(f'  Enter imaginary part of complex {i+1}: '))
        complex_list.append(Complex(r, im))

    total = complex_list[0]
    for i in range(1, N):
```

```
    total = add_complex(total, complex_list[i])

print('\nComplex Numbers entered:')
for idx, c in enumerate(complex_list, 1):
    print(f'  C{idx} = {c}')
print(f'\nSum of {N} complex numbers = {total}')
```

Sample Output

```
Enter number of complex numbers (N >= 2): 3
  Enter real part of complex 1     : 2
  Enter imaginary part of complex 1: 3
  Enter real part of complex 2     : 4
  Enter imaginary part of complex 2: -1
  Enter real part of complex 3     : 1
  Enter imaginary part of complex 3: 5

Complex Numbers entered:
  C1 = 2.0 + 3.0i
  C2 = 4.0 - 1.0i
  C3 = 1.0 + 5.0i

Sum of 3 complex numbers = 7.0 + 7.0i
```

Experiment 9: Text Analysis Tool

Aim	Build a tool that analyses a paragraph: frequency of each word, longest word, number of sentences, etc.
Objective	To apply string methods, dictionary operations, and build a comprehensive text analysis utility.

Program Code

```
# Experiment 9: Text Analysis Tool

import string

def analyse_text(text):
    # ── Sentence count ──
    sentences = [s.strip() for s in text.replace('!','.')
                                        .replace('?','.')
                                        .split('.') if s.strip()]
    num_sentences = len(sentences)

    # ── Word processing ──
    clean = text.lower()
    for ch in string.punctuation:
        clean = clean.replace(ch, '')
    words = clean.split()

    num_words   = len(words)
    num_chars   = len(text.replace(','))

    # ── Longest word ──
    longest = max(words, key=len) if words else ''

    # ── Word frequency ──
    freq = {}
    for w in words:
        freq[w] = freq.get(w, 0) + 1
    top5 = sorted(freq.items(), key=lambda x: x[1], reverse=True)[:5]

    # ── Report ──
    print('\n========== Text Analysis Report ==========')
    print(f'Number of sentences    : {num_sentences}')
```

```
    print(f'Number of words       : {num_words}')
    print(f'Number of characters  : {num_chars}')
    print(f'Unique words          : {len(freq)}')
    print(f'Longest word          : "{longest}" ({len(longest)} chars)')
    print(f'Average word length   : {sum(len(w) for w in words)/num_words:.2f}')
    print('\nTop 5 most frequent words:')
    for w, c in top5:
        print(f'  {w:<20} -> {c}')

paragraph = input('Enter a paragraph (press Enter twice to finish):\n>')
if paragraph:
    analyse_text(paragraph)
```

Sample Output

```
Enter a paragraph (press Enter twice to finish):
> Python is a powerful language. Python is easy to learn.
> Python supports object oriented programming. It is widely used.

========== Text Analysis Report ==========
Number of sentences    : 4
Number of words        : 17
Number of characters   : 82
Unique words           : 13
Longest word           : "programming" (11 chars)
Average word length    : 5.29

Top 5 most frequent words:
  python               -> 3
  is                   -> 3
  a                    -> 1
  powerful             -> 1
  language             -> 1
```

Experiment 10: Data Summary Generator from CSV File

Aim	Develop Data Summary Generator: Read a CSV file (like COVID data or weather stats), convert to dictionary form, and allow the user to run summary queries: max, min, average by column.
Objective	To practice file I/O with CSV format, dictionary construction from rows, and applying aggregate operations on columnar data.

Program Code

```
# Experiment 10: Data Summary Generator from CSV
# Sample CSV: data.csv
# date,cases,deaths,recoveries
# 2021-01-01,500,10,400
# 2021-01-02,620,15,510
# ... etc.

import csv

filename = input('Enter CSV file name: ')
try:
    with open(filename, newline='') as f:
        reader = csv.DictReader(f)
        rows = list(reader)
        headers = reader.fieldnames

    if not rows:
        print('CSV file is empty.')
    else:
        print(f'\nFile loaded: {filename}')
        print(f'Rows: {len(rows)}, Columns: {headers}')

        # Find numeric columns
        num_cols = []
        for col in headers:
            try:
                float(rows[0][col])
                num_cols.append(col)
            except (ValueError, TypeError):
                pass
```

```
        print('\n========== Summary Report ==========')
        print(f'{"Column":<20}{"Min":>10}{"Max":>10}{"Average":>12}')
        print('-' * 54)
        for col in num_cols:
            vals = [float(r[col]) for r in rows]

print(f'{col:<20}{min(vals):>10.2f}{max(vals):>10.2f}{sum(vals)/len(vals):>12.2f}')

        while True:
            q = input('\nQuery column (or "exit"): ')
            if q == 'exit': break
            if q in num_cols:
                vals = [float(r[q]) for r in rows]
                print(f'  Max={max(vals)}, Min={min(vals)},
Avg={sum(vals)/len(vals):.2f}')
            else:
                print('Column not found or not numeric.')

except FileNotFoundError:
    print(f'File "{filename}" not found!')
```

Sample Output

```
Enter CSV file name: covid_data.csv

File loaded: covid_data.csv
Rows: 7, Columns: ['date', 'cases', 'deaths', 'recoveries']

========== Summary Report ==========
Column                   Min       Max      Average
------------------------------------------------------
cases                 500.00    920.00       710.00
deaths                 10.00     22.00        15.71
recoveries            400.00    810.00       600.00

Query column (or "exit"): cases
  Max=920.0, Min=500.0, Avg=710.00
Query column (or "exit"): exit
```

Experiment 11: Student Grade Tracker

Aim	Develop Student Grade Tracker: Accept multiple students' names and marks. Store them in a list of tuples or dictionaries. Display summary reports (average, topper, etc.).
Objective	To practice list of dictionaries/tuples, aggregate computations, sorting by key, and formatted report generation.

▸ Program Code

```
# Experiment 11: Student Grade Tracker

SUBJECTS = ['Maths', 'Science', 'English', 'CS', 'Social']
MAX_MARKS = 100

def get_grade(percentage):
    if percentage >= 90: return 'O'
    elif percentage >= 80: return 'A+'
    elif percentage >= 70: return 'A'
    elif percentage >= 60: return 'B+'
    elif percentage >= 50: return 'B'
    elif percentage >= 40: return 'C'
    else: return 'F'

n = int(input('Enter number of students: '))
students = []

for i in range(n):
    print(f'\n--- Student {i+1} ---')
    name = input('  Name: ')
    marks = []
    for sub in SUBJECTS:
        m = int(input(f'  Marks in {sub}: '))
        marks.append(m)
    total = sum(marks)
    pct   = total / len(SUBJECTS)
    students.append({'name':name, 'marks':marks,
'total':total, 'pct':pct, 'grade':get_grade(pct)})

# Sort by total descending
students.sort(key=lambda s: s['total'], reverse=True)
```

```
print('\n============= Grade Report =============')
print(f'{"Rank":<5}{"Name":<15}{"Total":<8}{"Avg":<8}{"Grade"}')
print('-' * 45)
for rank, s in enumerate(students, 1):
    print(f'{rank:<5}{s["name"]:<15}{s["total"]:<8}{s["pct"]:<8.1f}{s["grade"]}')

# Summary
avg_total = sum(s['total'] for s in students) / n
topper    = students[0]
print('-' * 45)
print(f'Class Average : {avg_total:.2f}')
print(f'Topper        : {topper["name"]} ({topper["total"]} marks)')
passed = sum(1 for s in students if s['pct'] >= 40)
print(f'Pass Rate     : {passed}/{n} ({100*passed/n:.0f}%)')
```

Sample Output

```
Enter number of students: 3

--- Student 1 ---
 Name: Aisha
 Marks in Maths: 92, Science: 88, English: 76, CS: 95, Social: 80

--- Student 2 ---
 Name: Raj
 Marks in Maths: 70, Science: 65, English: 80, CS: 72, Social: 68

--- Student 3 ---
 Name: Priya
 Marks in Maths: 85, Science: 90, English: 88, CS: 91, Social: 87

============= Grade Report =============
Rank  Name         Total   Avg     Grade
---------------------------------------------
1     Aisha        431     86.2    A+
2     Priya        441     88.2    A+
3     Raj          355     71.0    A
---------------------------------------------
Class Average : 409.00
Topper        : Priya (441 marks)
```

Pass Rate : 3/3 (100%)

Experiment 12: Display Contents of a Directory Recursively

Aim	Develop a program to display contents of a folder recursively (Directory) having sub-folders and files (name and type).
Objective	To understand the os module, os.walk() for recursive directory traversal, and file type identification.

Program Code

```
# Experiment 12: Recursive Directory Listing

import os

def list_directory(path, indent=0):
    prefix = '' * indent
    try:
        entries = os.listdir(path)
        entries.sort()
        for entry in entries:
            full_path = os.path.join(path, entry)
            if os.path.isdir(full_path):
                print(f'{prefix}[DIR]  {entry}/')
                list_directory(full_path, indent + 1)
            else:
                size = os.path.getsize(full_path)
                ext  = os.path.splitext(entry)[1] or 'no ext'
                print(f'{prefix}[FILE] {entry:<30} Type: {ext:<10} Size: {size} bytes')
    except PermissionError:
        print(f'{prefix}[Permission Denied]')

# Alternative: using os.walk() for flat listing
def walk_directory(root):
    print(f'\n--- os.walk() listing of: {root} ---')
    for dirpath, dirnames, filenames in os.walk(root):
        level = dirpath.replace(root, '').count(os.sep)
        indent = '' * level
        print(f'{indent}[{os.path.basename(dirpath)}/]')
        subindent = '' * (level + 1)
        for f in filenames:
            print(f'{subindent}{f}')
```

```
folder = input('Enter folder path (or "." for current): ')
if os.path.isdir(folder):
    print(f'\n=== Recursive listing of: {os.path.abspath(folder)} ===')
    list_directory(folder)
    walk_directory(folder)
else:
    print('Invalid directory path!')
```

Sample Output

Enter folder path (or "." for current): ./project

```
=== Recursive listing of: /home/user/project ===
[DIR]  data/
  [FILE] covid_data.csv          Type: .csv      Size: 342 bytes
  [FILE] input.txt               Type: .txt      Size: 128 bytes
[DIR]  src/
  [FILE] exp1.py                 Type: .py       Size: 512 bytes
  [FILE] exp2.py                 Type: .py       Size: 480 bytes
  [DIR]  utils/
    [FILE] helper.py             Type: .py       Size: 256 bytes
[FILE] README.md                 Type: .md       Size: 1024 bytes

--- os.walk() listing of: ./project ---
[project/]
  README.md
  [data/]
    covid_data.csv
    input.txt
  [src/]
    exp1.py
    exp2.py
    [utils/]
      helper.py
```