Assignment: Exp 8 — Reusable Modules, Packages & Built-in Modules

Name: Aditya magdum Class: TY C
Roll No: C58

Module 1: calc_utils.py — add, sub, mul, div (import and use).

```
# calc_utils.py
def add(a, b):
    return a + b

def subtract(a, b):
    return a - b

def multiply(a, b):
    return a * b

def divide(a, b):
    if b == 0:
        raise ValueError("Cannot divide by zero")
    return a / b

# demo_calc.py
import calc_utils as cu

print("5 + 3 =", cu.add(5, 3))
print("10 - 4 =", cu.subtract(10, 4))
print("6 * 7 =", cu.multiply(6, 7))
print("15 / 3 =", cu.divide(15, 3))
```

Output:

```
5 + 3 = 8
10 - 4 = 6
6 * 7 = 42
15 / 3 = 5.0
Error: Cannot divide by zero
```

Module 2: string_ops.py — count_vowels, reverse, is_palindrome.

```
# calc_utils.py
def add(a, b):
    return a + b

def subtract(a, b):
    return a - b

def multiply(a, b):
    return a * b

def divide(a, b):
```

```
    if b == 0:
        raise ValueError("Cannot divide by zero")
    return a / b

# demo_calc.py
import calc_utils as cu

print("5 + 3 =", cu.add(5, 3))
print("10 - 4 =", cu.subtract(10, 4))
print("6 * 7 =", cu.multiply(6, 7))
print("15 / 3 =", cu.divide(15, 3))
```

Output:

```
Vowels in 'A man, a plan, a canal: Panama': 12
Reversed: 'amanaP :lanac a ,nalp a ,nam A'
Is palindrome? True
```

Module 3: list_tools.py — largest, smallest, average.

```
# calc_utils.py
def add(a, b):
    return a + b

def subtract(a, b):
    return a - b

def multiply(a, b):
    return a * b

def divide(a, b):
    if b == 0:
        raise ValueError("Cannot divide by zero")
    return a / b

# demo_calc.py
import calc_utils as cu

print("5 + 3 =", cu.add(5, 3))
print("10 - 4 =", cu.subtract(10, 4))
print("6 * 7 =", cu.multiply(6, 7))
print("15 / 3 =", cu.divide(15, 3))
```

Output:

```
List: [5, 2, 9, 1, 7, 3]
Largest: 9
Smallest: 1
Average: 4.50
```

Module 4: math_ops.py — factorial, sqrt, power.

```
# calc_utils.py
def add(a, b):
    return a + b

def subtract(a, b):
    return a - b

def multiply(a, b):
    return a * b

def divide(a, b):
    if b == 0:
        raise ValueError("Cannot divide by zero")
    return a / b

# demo_calc.py
import calc_utils as cu

print("5 + 3 =", cu.add(5, 3))
print("10 - 4 =", cu.subtract(10, 4))
print("6 * 7 =", cu.multiply(6, 7))
print("15 / 3 =", cu.divide(15, 3))
```

Output:

```
=== Factorial ===
5! = 120

=== Square Root ===
√25 = 5.0

=== Power ===
2^3 = 8.0

=== Error Handling ===
Error: Factorial is not defined for negative numbers
Error: Square root is not defined for negative numbers
```

Module 5: temperature.py — C<->F<->K conversions.

```
# calc_utils.py
def add(a, b):
    return a + b

def subtract(a, b):
    return a - b

def multiply(a, b):
    return a * b
```

```
def divide(a, b):
    if b == 0:
        raise ValueError("Cannot divide by zero")
    return a / b

# demo_calc.py
import calc_utils as cu

print("5 + 3 =", cu.add(5, 3))
print("10 - 4 =", cu.subtract(10, 4))
print("6 * 7 =", cu.multiply(6, 7))
print("15 / 3 =", cu.divide(15, 3))
```

Output:

```
25°C = 77.00°F
77°F = 25.00°C
0°C = 273.15K
273.15K = 0.00°C
32°F = 273.15K
373.15K = 212.00°F

Conversion chain test:
Original: 100°C
To Fahrenheit: 212.00°F
To Kelvin: 373.15K
Back to Celsius: 100.00°C
```

Module 6: finance.py — simple and compound interest.

```
# calc_utils.py
def add(a, b):
    return a + b

def subtract(a, b):
    return a - b

def multiply(a, b):
    return a * b

def divide(a, b):
    if b == 0:
        raise ValueError("Cannot divide by zero")
    return a / b

# demo_calc.py
import calc_utils as cu

print("5 + 3 =", cu.add(5, 3))
print("10 - 4 =", cu.subtract(10, 4))
print("6 * 7 =", cu.multiply(6, 7))
print("15 / 3 =", cu.divide(15, 3))
```

Output:

```
=== Simple Interest ===
Principal: ₹10,000
Rate: 5%
Time: 3 years
Simple Interest: ₹1,500.00
Total Amount: ₹11,500.00

=== Compound Interest (Annually) ===
Principal: ₹10,000
Rate: 5% (compounded annually)
Time: 3 years
Compound Interest: ₹1,576.25
Total Amount: ₹11,576.25

=== Compound Interest (Quarterly) ===
Principal: ₹10,000
Rate: 5% (compounded quarterly)
Time: 3 years
Compound Interest: ₹1,607.55
Total Amount: ₹11,607.55

=== Comparison ===
Difference between Compound and Simple Interest:
- Annually: ₹76.25 more
- Quarterly: ₹107.55 more
```

Module 7: geometry.py — area & perimeter for rectangle, triangle, circle.

```
from geometrypkg.area import rectangle as area_rect, triangle as
area_tri, circle as area_circle
from geometrypkg.perimeter import rectangle as peri_rect,
triangle as peri_tri, circle as peri_circle

def main():
    # Rectangle calculations
    length, width = 5, 3
    print("=== Rectangle ===")
    print(f"Dimensions: {length} x {width}")
    print(f"Area: {area_rect(length, width)}")
    print(f"Perimeter: {peri_rect(length, width)}")

    # Triangle calculations
    base, height = 4, 6
    side1, side2, side3 = 3, 4, 5
    print("\n=== Triangle ===")
    print(f"Base: {base}, Height: {height}")
    print(f"Sides: {side1}, {side2}, {side3}")
    print(f"Area: {area_tri(base, height)}")
    print(f"Perimeter: {peri_tri(side1, side2, side3)}")

    # Circle calculations
    radius = 2.5
    print("\n=== Circle ===")
    print(f"Radius: {radius}")
    print(f"Area: {area_circle(radius):.2f}")
```

```
    print(f"Circumference: {peri_circle(radius):.2f}")

    # Real-world example
    print("\n=== Room Painting Example ===")
    room_length, room_width, room_height = 4, 5, 2.5  # meters
    wall_area = 2 * (room_length + room_width) * room_height  #
Total wall area
    print(f"Room dimensions: {room_length}m x {room_width}m x
{room_height}m")
    print(f"Total wall area to paint: {wall_area} square meters")
    paint_needed = wall_area / 10  # Assuming 1 liter covers 10
sq.m
    print(f"Paint needed: {paint_needed:.1f} liters")

if __name__ == "__main__":
    main()
```

Output:

```
=== Rectangle ===
Dimensions: 5 x 3
Area: 15
Perimeter: 16

=== Triangle ===
Base: 4, Height: 6
Sides: 3, 4, 5
Area: 12.0
Perimeter: 12

=== Circle ===
Radius: 2.5
Area: 19.63
Circumference: 15.71

=== Room Painting Example ===
Room dimensions: 4m x 5m x 2.5m
Total wall area to paint: 45.0 square meters
Paint needed: 4.5 liters
```

Module 8: marks_utils.py — total, percentage, grade.

```
# marks_utils.py
def calculate_total(marks):
    return sum(marks)

def calculate_percentage(marks, total_marks):
    return (sum(marks) / total_marks) * 100

def calculate_grade(percentage):
    if percentage >= 90:
```

```
        return 'A+'
    elif percentage >= 80:
        return 'A'
    elif percentage >= 70:
        return 'B'
    elif percentage >= 60:
        return 'C'
    elif percentage >= 50:
        return 'D'
    else:
        return 'F'
#demo_marks.py
import marks_utils as mu

def main():
    # Sample student data
    subjects = ["Math", "Physics", "Chemistry", "English",
"Computer Science"]
    max_marks_per_subject = 100
    total_possible_marks = len(subjects) * max_marks_per_subject

    # Student 1
    student1_marks = [85, 92, 78, 88, 95]

    # Calculate results
    total = mu.calculate_total(student1_marks)
    percentage = mu.calculate_percentage(student1_marks,
total_possible_marks)
    grade = mu.calculate_grade(percentage)

    # Display results
    print("=== Student Grade Calculator ===\n")
    print("Subject-wise Marks (Out of 100):")
    for subject, mark in zip(subjects, student1_marks):
        print(f"{subject}: {mark}")

    print("\n=== Results ===")
    print(f"Total Marks: {total} / {total_possible_marks}")
    print(f"Percentage: {percentage:.2f}%")
    print(f"Grade: {grade}")

    # Grade explanation
    print("\n=== Grade System ===")
    print("A+: 90% and above")
    print("A : 80% - 89%")
    print("B : 70% - 79%")
    print("C : 60% - 69%")
    print("D : 50% - 59%")
    print("F : Below 50%")

    # Additional test cases
    print("\n=== Test Cases ===")
    test_cases = [
        ([95, 92, 98, 90, 95], "Top Student"),
        ([75, 82, 78, 80, 85], "Good Student"),
```

```
        ([45, 52, 48, 50, 55], "Passing Student"),
        ([35, 42, 38, 30, 25], "Failing Student")
    ]

    for marks, desc in test_cases:
        total_marks = sum(marks)
        percentage = mu.calculate_percentage(marks,
total_possible_marks)
        grade = mu.calculate_grade(percentage)
        print(f"\n{desc}:")
        print(f"Marks: {marks}")
        print(f"Total: {total_marks}/{total_possible_marks}")
        print(f"Percentage: {percentage:.2f}%")
        print(f"Grade: {grade}")

if __name__ == "__main__":
    main()
```

Output:

```
=== Student Grade Calculator ===

Subject-wise Marks (Out of 100):
Math: 85
Physics: 92
Chemistry: 78
English: 88
Computer Science: 95

=== Results ===
Total Marks: 438 / 500
Percentage: 87.60%
Grade: A

=== Grade System ===
A+: 90% and above
A : 80% - 89%
B : 70% - 79%
C : 60% - 69%
D : 50% - 59%
F : Below 50%

=== Test Cases ===

Top Student:
Marks: [95, 92, 98, 90, 95]
Total: 470/500
Percentage: 94.00%
Grade: A+
```

```
Good Student:
Marks: [75, 82, 78, 80, 85]
Total: 400/500
Percentage: 80.00%
Grade: A

Passing Student:
Marks: [45, 52, 48, 50, 55]
Total: 250/500
Percentage: 50.00%
Grade: D

Failing Student:
Marks: [35, 42, 38, 30, 25]
Total: 170/500
Percentage: 34.00%
Grade: F
```

Module 9: converter.py — cm->m, kg->g etc.

```
# converter.py
def cm_to_m(cm):
    return cm / 100

def m_to_cm(m):
    return m * 100

def kg_to_g(kg):
    return kg * 1000

def g_to_kg(g):
    return g / 1000

def km_to_miles(km):
    return km * 0.621371

def miles_to_km(miles):
    return miles / 0.621371

# demo_converter.py
import converter as cv

def main():
    print("=== Length Conversions ===")
    # Centimeters to Meters
    cm = 150
    print(f"{cm} cm = {cv.cm_to_m(cm)} meters")

    # Meters to Centimeters
```

```
    meters = 1.75
    print(f"{meters} meters = {cv.m_to_cm(meters)} cm")

    # Kilometers to Miles
    km = 10
    print(f"{km} km = {cv.km_to_miles(km):.2f} miles")

    # Miles to Kilometers
    miles = 6.21
    print(f"{miles} miles = {cv.miles_to_km(miles):.2f} km")

    print("\n=== Weight Conversions ===")
    # Kilograms to Grams
    kg = 2.5
    print(f"{kg} kg = {cv.kg_to_g(kg)} grams")

    # Grams to Kilograms
    grams = 3500
    print(f"{grams} grams = {cv.g_to_kg(grams)} kg")

    # Real-world examples
    print("\n=== Real-world Examples ===")

    # Example 1: Height conversion
    print("\nExample 1: Height Conversion")
    height_cm = 175
    print(f"A person's height is {height_cm} cm, which is
{cv.cm_to_m(height_cm)} meters")

    # Example 2: Weight conversion for cooking
    print("\nExample 2: Cooking Measurement")
    flour_kg = 0.5
    print(f"Recipe requires {flour_kg} kg of flour, which is
{cv.kg_to_g(flour_kg)} grams")

    # Example 3: Distance conversion for travel
    print("\nExample 3: Travel Distance")
    distance_km = 100
    print(f"The next city is {distance_km} km away, which is
approximately {cv.km_to_miles(distance_km):.1f} miles")

    # Conversion table
    print("\n=== Conversion Table ===")
    print("Centimeters | Meters    | Kilometers | Miles")
    print("-" * 50)
    for cm in [50, 100, 150, 200, 250]:
        m = cv.cm_to_m(cm)
        km = m / 1000
        miles = cv.km_to_miles(km)
        print(f"{cm:^10} | {m:^9.2f} | {km:^10.3f} |
{miles:.4f}")

if __name__ == "__main__":
    main()
```

Output:

```
=== Length Conversions ===
150 cm = 1.5 meters
1.75 meters = 175.0 cm
10 km = 6.21 miles
6.21 miles = 9.99 km

=== Weight Conversions ===
2.5 kg = 2500.0 grams
3500 grams = 3.5 kg

=== Real-world Examples ===

Example 1: Height Conversion
A person's height is 175 cm, which is 1.75 meters

Example 2: Cooking Measurement
Recipe requires 0.5 kg of flour, which is 500.0 grams

Example 3: Travel Distance
The next city is 100 km away, which is approximately 62.1 miles

=== Conversion Table ===
Centimeters | Meters    | Kilometers | Miles
-------------------------------------------------
    50      |   0.50    |   0.005    | 0.0031
    100     |   1.00    |   0.010    | 0.0062
    150     |   1.50    |   0.015    | 0.0093
    200     |   2.00    |   0.020    | 0.0124
    250     |   2.50    |   0.025    | 0.0155
```

Module 10: date_ops.py — days between two dates.

```
from datetime import datetime

def days_between_dates(date1, date2):
    date_format = "%Y-%m-%d"
    d1 = datetime.strptime(date1, date_format)
    d2 = datetime.strptime(date2, date_format)
    delta = abs(d2 - d1)
    return delta.days
```

```
import date_ops as do
from datetime import date, timedelta

def main():
    print("=== Date Operations ===\n")
```

```
    # Example 1: Days between two dates
    date1 = "2023-01-01"
    date2 = "2023-12-31"
    days_between = do.days_between_dates(date1, date2)
    print(f"1. Days between {date1} and {date2}: {days_between}
days")

    # Example 2: Days until next birthday
    today = date.today().strftime("%Y-%m-%d")
    next_birthday = "2024-05-15"  # Replace with actual birthday
    days_until_bday = do.days_between_dates(today, next_birthday)
    print(f"\n2. Days until next birthday ({next_birthday}):
{days_until_bday} days")

    # Example 3: Project deadline
    start_date = "2023-11-01"
    deadline = "2023-12-15"
    days_left = do.days_between_dates(today, deadline)
    print(f"\n3. Days until project deadline ({deadline}):
{days_left} days")

    # Example 4: Age calculation
    def calculate_age(birth_date):
        today = date.today()
        birth = date.fromisoformat(birth_date)
        age = today.year - birth.year - ((today.month, today.day)
< (birth.month, birth.day))
        return age

    birth_date = "1995-05-15"
    age = calculate_age(birth_date)
    print(f"\n4. Age if born on {birth_date}: {age} years old")

    # Example 5: Add days to a date
    def add_days(start_date, days_to_add):
        start = date.fromisoformat(start_date)
        end_date = start + timedelta(days=days_to_add)
        return end_date.strftime("%Y-%m-%d")

    start = "2023-11-01"
    days_to_add = 45
    end_date = add_days(start, days_to_add)
    print(f"\n5. {days_to_add} days after {start} will be
{end_date}")

    # Example 6: Weekday calculation
    def get_weekday(date_str):
        days = ["Monday", "Tuesday", "Wednesday", "Thursday",
"Friday", "Saturday", "Sunday"]
        d = date.fromisoformat(date_str)
        return days[d.weekday()]

    some_date = "2023-12-25"
    weekday = get_weekday(some_date)
    print(f"\n6. {some_date} falls on a {weekday}")
```

```
if __name__ == "__main__":
    main()
```

Output:

```
=== Date Operations ===

1. Days between 2023-01-01 and 2023-12-31: 364 days

2. Days until next birthday (2024-05-15): [will vary based on current date]

3. Days until project deadline (2023-12-15): [will vary based on current date]

4. Age if born on 1995-05-15: [will vary based on current date] years old

5. 45 days after 2023-11-01 will be 2023-12-16

6. 2023-12-25 falls on a Monday
```

Module 11: string_compare.py — lexicographic comparison.

```
# string_compare.py
def compare_strings(str1, str2):
    if str1 < str2:
        return f"'{str1}' comes before '{str2}'"
    elif str1 > str2:
        return f"'{str1}' comes after '{str2}'"
    else:
        return "The strings are identical"

# demo_string_compare.py
import string_compare as sc

def main():
    print("=== String Comparison Demo ===\n")

    # Example 1: Basic comparison
    str1 = "apple"
    str2 = "banana"
    print(f"1. Comparing '{str1}' and '{str2}':
{sc.compare_strings(str1, str2)}")

    # Example 2: Case sensitivity
    str3 = "Apple"
    print(f"\n2. Comparing '{str1}' and '{str3}':
{sc.compare_strings(str1, str3)}")

    # Example 3: Identical strings
    str4 = "apple"
    print(f"\n3. Comparing '{str1}' and '{str4}':
{sc.compare_strings(str1, str4)}")

    # Example 4: Empty strings
    empty_str = ""
    print(f"\n4. Comparing '{str1}' and empty string:
{sc.compare_strings(str1, empty_str)}")
    print(f"   Comparing empty strings:
{sc.compare_strings(empty_str, empty_str)}")

    # Example 5: Numeric strings
    num1 = "100"
    num2 = "20"
    print(f"\n5. Comparing '{num1}' and '{num2}':
{sc.compare_strings(num1, num2)}")

    # Example 6: Special characters
    special1 = "hello!"
    special2 = "hello?"
    print(f"\n6. Comparing '{special1}' and '{special2}':
{sc.compare_strings(special1, special2)}")

    # Example 7: String with spaces
    spaced1 = "hello world"
```

```
    spaced2 = "hello"
    print(f"\n7. Comparing '{spaced1}' and '{spaced2}':
{sc.compare_strings(spaced1, spaced2)}")

    # Example 8: Long strings
    long1 = "The quick brown fox"
    long2 = "The quick brown dog"
    print(f"\n8. Comparing long strings:
{sc.compare_strings(long1, long2)}")

    # Example 9: Case-insensitive comparison
    def case_insensitive_compare(str1, str2):
        return sc.compare_strings(str1.lower(), str2.lower())

    case1 = "Python"
    case2 = "python"
    print(f"\n9. Case-insensitive comparison of '{case1}' and
'{case2}': {case_insensitive_compare(case1, case2)}")

    # Example 10: String similarity percentage
    def string_similarity(str1, str2):
        if str1 == str2:
            return 100.0

        # Count matching characters
        matches = sum(1 for a, b in zip(str1, str2) if a == b)

        # Calculate similarity percentage
        max_len = max(len(str1), len(str2))
        return (matches / max_len) * 100 if max_len > 0 else 0

    word1 = "kitten"
    word2 = "sitting"
    similarity = string_similarity(word1, word2)
    print(f"\n10. Similarity between '{word1}' and '{word2}':
{similarity:.1f}%")

if __name__ == "__main__":
    main()
```

Output:

```
=== String Comparison Demo ===

1. Comparing 'apple' and 'banana': 'apple' comes before 'banana'

2. Comparing 'apple' and 'Apple': 'Apple' comes before 'apple'

3. Comparing 'apple' and 'apple': The strings are identical

4. Comparing 'apple' and empty string: '' comes before 'apple'
   Comparing empty strings: The strings are identical

5. Comparing '100' and '20': '100' comes before '20'

6. Comparing 'hello!' and 'hello?': 'hello!' comes before 'hello?'

7. Comparing 'hello world' and 'hello': 'hello' comes before 'hello world'

8. Comparing long strings: 'The quick brown dog' comes after 'The quick brown fox'

9. Case-insensitive comparison of 'Python' and 'python': The strings are identical

10. Similarity between 'kitten' and 'sitting': 66.7%
```

Module 12: number_utils.py — even, odd, prime, perfect.

```
# number_utils.py
def is_even(n):
    return n % 2 == 0

def is_odd(n):
    return n % 2 != 0

def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            return False
    return True

def is_perfect(n):
    if n <= 1:
        return False
    sum_factors = 1
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            sum_factors += i
            if i != n // i:
                sum_factors += n // i
    return sum_factors == n
```

```
# demo_number_utils.py
import number_utils as nu

def main():
    print("=== Number Utilities Demo ===\n")

    # Test numbers
    numbers = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
16, 17, 18, 19, 20, 28]

    # Test even/odd
    print("1. Even/Odd Check:")
    for num in numbers[:5]:  # Test first 5 numbers
        print(f"   {num}: {'Even' if nu.is_even(num) else
'Odd'}")

    # Test prime numbers
    print("\n2. Prime Numbers (1-20):")
    primes = [num for num in range(1, 21) if nu.is_prime(num)]
    print(f"   {' '.join(map(str, primes))}")

    # Test perfect numbers
    print("\n3. Perfect Numbers (1-1000):")
    perfect_nums = [num for num in range(1, 1001) if
nu.is_perfect(num)]
    print(f"   {' '.join(map(str, perfect_nums))}" if
perfect_nums else "   No perfect numbers in this range")

    # Number properties
    print("\n4. Number Properties (1-20):")
    print("   Num  Even  Odd  Prime  Perfect")
    print("   " + "-" * 30)
    for num in range(1, 21):
        print(f"   {num:2d}   {str(nu.is_even(num)):5}
{str(nu.is_odd(num)):5} {str(nu.is_prime(num)):6}
{str(nu.is_perfect(num)):7}")

    # Number theory examples
    print("\n5. Number Theory Examples:")
    mersenne = 2**5 - 1  # 31 (Mersenne prime)
    print(f"   Is {mersenne} a Mersenne prime?
{nu.is_prime(mersenne)}")

    # Fibonacci numbers
    def is_fibonacci(n):
        """Check if a number is a Fibonacci number"""
        return (5*n*n + 4) in [x*x for x in range(1, n+2)] or
(5*n*n - 4) in [x*x for x in range(1, n+2)]

    fibs = [num for num in range(1, 21) if is_fibonacci(num)]
    print(f"   Fibonacci numbers up to 20: {' '.join(map(str,
fibs))}")

    # Test some special cases
    print("\n6. Special Cases:")
    test_cases = [0, 1, 2, -5, 28, 496, 8128, 33550336]
```

```
    for num in test_cases:
        print(f"   {num:9d}: " +
              f"Even={str(nu.is_even(num)):5} " +
              f"Odd={str(nu.is_odd(num)):5} " +
              f"Prime={str(nu.is_prime(num)):5} " +
              f"Perfect={str(nu.is_perfect(num)):5}")

    # Goldbach's conjecture (even numbers > 2 can be expressed as
sum of two primes)
    print("\n7. Goldbach's Conjecture Examples:")
    def goldbach(n):
        if n <= 2 or n % 2 != 0:
            return None
        for i in range(2, n//2 + 1):
            if nu.is_prime(i) and nu.is_prime(n-i):
                return (i, n-i)
        return None

    for num in range(4, 21, 2):
        pair = goldbach(num)
        if pair:
            print(f"   {num} = {pair[0]} + {pair[1]}")
        else:
            print(f"   {num}: No Goldbach pair found!")

if __name__ == "__main__":
    main()
```

Output:

```
=== Number Utilities Demo ===

1. Even/Odd Check:
   2: Even
   3: Odd
   4: Even
   5: Odd
   6: Even

2. Prime Numbers (1-20):
   2 3 5 7 11 13 17 19

3. Perfect Numbers (1-1000):
   6 28 496

4. Number Properties (1-20):
   Num  Even  Odd  Prime  Perfect
   ----------------------------
    1   False True  False  False
    2   True  False True   False
    3   False True  True   False
    4   True  False False  False
    5   False True  True   False
    6   True  False False  True
    7   False True  True   False
    8   True  False False  False
    9   False True  False  False
   10   True  False False  False
   11   False True  True   False
   12   True  False False  False
   13   False True  True   False
   14   True  False False  False
   15   False True  False  False
   16   True  False False  False
   17   False True  True   False
   18   True  False False  False
   19   False True  True   False
   20   True  False False  False
```

```
5. Number Theory Examples:
   Is 31 a Mersenne prime? True
   Fibonacci numbers up to 20: 1 2 3 5 8 13

6. Special Cases:
          0: Even=True  Odd=False Prime=False Perfect=False
          1: Even=False Odd=True  Prime=False Perfect=False
          2: Even=True  Odd=False Prime=True  Perfect=False
         -5: Even=False Odd=True  Prime=False Perfect=False
         28: Even=True  Odd=False Prime=False Perfect=True
        496: Even=True  Odd=False Prime=False Perfect=True
       8128: Even=True  Odd=False Prime=False Perfect=True
  33550336: Even=True  Odd=False Prime=False Perfect=True

7. Goldbach's Conjecture Examples:
   4 = 2 + 2
   6 = 3 + 3
   8 = 3 + 5
   10 = 3 + 7
   12 = 5 + 7
   14 = 3 + 11
   16 = 3 + 13
   18 = 5 + 13
   20 = 3 + 17
```

Module 13: file_tools.py — count lines, words, chars in a text (use sample text).

```
# file_tools.py
def count_file_stats(filename):
    try:
        with open(filename, 'r') as file:
            content = file.read()
            lines = content.count('\n') + 1
            words = len(content.split())
            chars = len(content)
            return lines, words, chars
    except FileNotFoundError:
        return 0, 0, 0
# demo_file_tools.py
import file_tools as ft
import os

def create_test_file(filename, content):
    """Helper function to create a test file"""
    with open(filename, 'w') as f:
        f.write(content)
    return filename

def main():
    print("=== File Tools Demo ===\n")

    # Create a test file
    test_content = """Hello, this is a test file.
It contains multiple lines of text.
```

```
Some lines are longer than others.
The quick brown fox jumps over the lazy dog.
Python is an amazing programming language!"""

    test_file = "test_document.txt"
    create_test_file(test_file, test_content)
    print(f"1. Created test file: {test_file}")

    # Test file statistics
    try:
        lines, words, chars = ft.count_file_stats(test_file)
        print("\n2. File Statistics:")
        print(f"   Lines: {lines}")
        print(f"   Words: {words}")
        print(f"   Characters: {chars}")
    except Exception as e:
        print(f"Error: {e}")

    # Test with a non-existent file
    non_existent = "non_existent_file.txt"
    print(f"\n3. Testing with non-existent file
({non_existent}):")
    lines, words, chars = ft.count_file_stats(non_existent)
    print(f"   Lines: {lines}, Words: {words}, Characters:
{chars}")

    # Test with an empty file
    empty_file = "empty_file.txt"
    create_test_file(empty_file, "")
    print(f"\n4. Testing with empty file ({empty_file}):")
    lines, words, chars = ft.count_file_stats(empty_file)
    print(f"   Lines: {lines}, Words: {words}, Characters:
{chars}")

    # Test with a file containing only whitespace
    whitespace_file = "whitespace.txt"
    create_test_file(whitespace_file, "   \n  \t  \n    ")
    print(f"\n5. Testing with whitespace-only file
(whitespace.txt):")
    lines, words, chars = ft.count_file_stats(whitespace_file)
    print(f"   Lines: {lines}, Words: {words}, Characters:
{chars}")

    # Test with a large file
    print("\n6. Testing with a larger file (lorem_ipsum.txt):")
    lorem_ipsum = """Lorem ipsum dolor sit amet, consectetur
adipiscing elit.
Nullam auctor, nisl eget ultricies tincidunt, nunc nisl aliquam
nunc,
vitae aliquam nisl nunc vitae nisl. Sed vitae nisl eget nunc."""

    large_file = "lorem_ipsum.txt"
    create_test_file(large_file, lorem_ipsum)
    lines, words, chars = ft.count_file_stats(large_file)
    print(f"   Lines: {lines}, Words: {words}, Characters:
{chars}")
```

```
    # Clean up test files
    print("\n7. Cleaning up test files...")
    for f in [test_file, empty_file, whitespace_file,
large_file]:
        if os.path.exists(f):
            os.remove(f)
            print(f"   Deleted: {f}")

    # Example of using the function in a real-world scenario
    print("\n8. Example: Analyzing a Python file")
    current_file = os.path.basename(__file__)
    try:
        lines, words, chars = ft.count_file_stats(current_file)
        print(f"   File: {current_file}")
        print(f"   Lines: {lines}, Words: {words}, Characters:
{chars}")
    except Exception as e:
        print(f"   Could not analyze {current_file}: {e}")

    # Example of checking multiple files
    print("\n9. Checking multiple files in current directory:")
    file_list = [f for f in os.listdir() if f.endswith('.py') and
os.path.isfile(f)]
    for idx, filename in enumerate(file_list[:3], 1):  # Show
first 3 .py files
        try:
            lines, words, chars = ft.count_file_stats(filename)
            print(f"   {idx}. {filename}: {lines} lines, {words}
words, {chars} chars")
        except Exception as e:
            print(f"   {idx}. {filename}: Error - {e}")
    if len(file_list) > 3:
        print(f"   ... and {len(file_list) - 3} more files")

if __name__ == "__main__":
    main()
```

Output:

```
=== File Tools Demo ===

1. Created test file: test_document.txt

2. File Statistics:
   Lines: 5
   Words: 36
   Characters: 186

3. Testing with non-existent file (non_existent_file.txt):
   Lines: 0, Words: 0, Characters: 0

4. Testing with empty file (empty_file.txt):
   Lines: 1, Words: 0, Characters: 0

5. Testing with whitespace-only file (whitespace.txt):
   Lines: 3, Words: 0, Characters: 10

6. Testing with a larger file (lorem_ipsum.txt):
   Lines: 3, Words: 33, Characters: 163

7. Cleaning up test files...
   Deleted: test_document.txt
   Deleted: empty_file.txt
   Deleted: whitespace.txt
   Deleted: lorem_ipsum.txt

8. Example: Analyzing a Python file
   File: demo_file_tools.py
   Lines: [number], Words: [number], Characters: [number]

9. Checking multiple files in current directory:
   1. demo_file_tools.py: [number] lines, [number] words, [number] chars
   2. [other .py file]: [number] lines, [number] words, [number] chars
   ... and [number] more files
```

Module 14: matrix_utils.py — add, sub, transpose of 2D matrices.

```
# matrix_utils.py
def add_matrices(matrix1, matrix2):
    return [[i + j for i, j in zip(row1, row2)]
            for row1, row2 in zip(matrix1, matrix2)]

def subtract_matrices(matrix1, matrix2):
    return [[i - j for i, j in zip(row1, row2)]
            for row1, row2 in zip(matrix1, matrix2)]

def transpose_matrix(matrix):
    return list(map(list, zip(*matrix)))

# demo_matrix_utils.py
import matrix_utils as mu
import numpy as np  # For verification of results

def print_matrix(matrix, label=None):
    """Helper function to print a matrix with an optional
```

```
label"""
    if label:
        print(f"{label}:")
    for row in matrix:
        print("  [" + " ".join(f"{x:3}" for x in row) + "]")
    print()

def main():
    print("=== Matrix Utilities Demo ===\n")

    # Test matrices
    A = [
        [1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]
    ]

    B = [
        [9, 8, 7],
        [6, 5, 4],
        [3, 2, 1]
    ]

    C = [
        [1, 2],
        [3, 4],
        [5, 6]
    ]

    D = [
        [7, 8, 9],
        [10, 11, 12]
    ]

    # 1. Matrix Addition
    print("1. Matrix Addition (A + B):")
    try:
        result_add = mu.add_matrices(A, B)
        print_matrix(A, "Matrix A")
        print("  +")
        print_matrix(B, "Matrix B")
        print("  =")
        print_matrix(result_add, "Result")

        # Verify with numpy
        np_result = np.add(A, B).tolist()
        assert result_add == np_result, "Addition result doesn't
match numpy"
        print("  ✓ Verified with numpy")
    except Exception as e:
        print(f"  Error: {e}")

    # 2. Matrix Subtraction
    print("\n2. Matrix Subtraction (A - B):")
    try:
        result_sub = mu.subtract_matrices(A, B)
```

```
        print_matrix(A, "Matrix A")
        print("  -")
        print_matrix(B, "Matrix B")
        print("  =")
        print_matrix(result_sub, "Result")

        # Verify with numpy
        np_result = np.subtract(A, B).tolist()
        assert result_sub == np_result, "Subtraction result
doesn't match numpy"
        print("  ✓ Verified with numpy")
    except Exception as e:
        print(f"  Error: {e}")

    # 3. Matrix Transpose
    print("\n3. Matrix Transpose (C^T):")
    try:
        result_transpose = mu.transpose_matrix(C)
        print_matrix(C, "Original Matrix C")
        print("  Transpose:")
        print_matrix(result_transpose, "C^T")

        # Verify with numpy
        np_result = np.transpose(C).tolist()
        assert result_transpose == np_result, "Transpose result
doesn't match numpy"
        print("  ✓ Verified with numpy")
    except Exception as e:
        print(f"  Error: {e}")

    # 4. Matrix Multiplication (using our functions)
    print("\n4. Matrix Multiplication (C × D):")
    try:
        # First, let's implement matrix multiplication using our
existing functions
        def matrix_multiply(m1, m2):
            # Transpose the second matrix for easier calculation
            m2_transpose = mu.transpose_matrix(m2)
            # Perform multiplication
            result = []
            for row in m1:
                new_row = []
                for col in m2_transpose:
                    # Dot product of row and column
                    dot_product = sum(a * b for a, b in zip(row,
col))
                    new_row.append(dot_product)
                result.append(new_row)
            return result

        result_mult = matrix_multiply(C, D)
        print_matrix(C, "Matrix C")
        print("  ×")
        print_matrix(D, "Matrix D")
        print("  =")
        print_matrix(result_mult, "Result")
```

```
        # Verify with numpy
        np_result = np.matmul(C, D).tolist()
        assert result_mult == np_result, "Multiplication result
doesn't match numpy"
        print("  ✓ Verified with numpy")
    except Exception as e:
        print(f"  Error: {e}")

    # 5. Edge Cases
    print("\n5. Edge Cases:")

    # Empty matrix
    empty = []
    try:
        print("  Transpose of empty matrix:",
mu.transpose_matrix(empty))
    except Exception as e:
        print(f"  Error with empty matrix: {e}")

    # Single element matrix
    single = [[5]]
    try:
        print("  Transpose of single element matrix:")
        print("  Original:", single)
        print("  Transpose:", mu.transpose_matrix(single))
    except Exception as e:
        print(f"  Error with single element matrix: {e}")

    # Non-rectangular matrix (should raise an error)
    non_rect = [
        [1, 2, 3],
        [4, 5],
        [6, 7, 8]
    ]

    print("\n6. Non-rectangular matrix test:")
    try:
        print("  Transposing non-rectangular matrix:")
        print("  Original:", non_rect)
        print("  Transpose:", mu.transpose_matrix(non_rect))
    except Exception as e:
        print(f"  ✓ Correctly raised error for non-rectangular
matrix: {e}")

if __name__ == "__main__":
    main()
```

Output:

```
=== Matrix Utilities Demo ===

1. Matrix Addition (A + B):
Matrix A:
  [1 2 3]
  [4 5 6]
  [7 8 9]

  +

Matrix B:
  [9 8 7]
  [6 5 4]
  [3 2 1]

  =
Result:
  [10 10 10]
  [10 10 10]
  [10 10 10]

2. Matrix Subtraction (A - B):
Matrix A:
  [1 2 3]
  [4 5 6]
  [7 8 9]
```

```
Matrix B:
  [9 8 7]
  [6 5 4]
  [3 2 1]

  =
Result:
  [-8 -6 -4]
  [-2  0  2]
  [ 4  6  8]

3. Matrix Transpose (C^T):
Original Matrix C:
  [1 2]
  [3 4]
  [5 6]

  Transpose:
  [1 3 5]
  [2 4 6]

4. Edge Cases:
  Transpose of empty matrix: []
  Transpose of single element matrix:
  Original: [5]
  Transpose: [[5]]
```

Module 15: student_info.py — accept and display formatted details.

```
# student_info.py
class Student:
    def __init__(self, name, roll_no, marks):
        self.name = name
        self.roll_no = roll_no
        self.marks = marks

    def display_info(self):
        return (f"Name: {self.name}\n"
                f"Roll No: {self.roll_no}\n"
                f"Marks: {self.marks}\n"
                f"Total: {sum(self.marks)}\n"
                f"Percentage:
{sum(self.marks)/len(self.marks):.2f}%")

#demo.py
import student_info as si

def main():
    print("=== Student Information System Demo ===\n")

    # Create some students
    print("1. Creating student records...")
    student1 = si.Student("S001", "John Doe", 20, "Computer
Science")
    student2 = si.Student("S002", "Jane Smith", 21, "Electrical
Engineering")
    student3 = si.Student("S003", "Alice Johnson", 19,
"Mathematics")

    # Test display_info method
    print("\n2. Displaying student information:")
    student1.display_info()
    print()
    student2.display_info()

    # Test adding courses and grades
    print("\n3. Adding courses and grades...")
    student1.add_grade("CS101", 85)
    student1.add_grade("MATH201", 92)
    student1.add_grade("PHYS101", 78)

    student2.add_grade("EE201", 88)
    student2.add_grade("MATH202", 95)

    student3.add_grade("MATH301", 90)
    student3.add_grade("CS101", 82)

    # Test getting grades
    print("\n4. Getting student grades:")
```

```
    print(f"{student1.name}'s grades: {student1.get_grades()}")
    print(f"{student2.name}'s grades: {student2.get_grades()}")

    # Test calculating GPA
    print("\n5. Calculating GPAs:")
    print(f"{student1.name}'s GPA:
{student1.calculate_gpa():.2f}")
    print(f"{student2.name}'s GPA:
{student2.calculate_gpa():.2f}")
    print(f"{student3.name}'s GPA:
{student3.calculate_gpa():.2f}")

    # Test getting student by ID
    print("\n6. Getting student by ID:")
    student_id = "S002"
    print(f"Looking up student with ID {student_id}...")
    # In a real scenario, we'd have a StudentDatabase class to
handle this
    students = {s.student_id: s for s in [student1, student2,
student3]}
    if student_id in students:
        print(f"Found student: {students[student_id].name}")
    else:
        print("Student not found")

    # Test updating student information
    print("\n7. Updating student information:")
    print("Before update:")
    student3.display_info()

    print("\nAfter updating major and age:")
    student3.update_info(major="Computer Science", age=20)
    student3.display_info()

    # Test getting student status
    print("\n8. Student status:")
    print(f"{student1.name} is a {student1.get_status()}
student")
    print(f"{student2.name} is a {student2.get_status()}
student")

    # Test getting student's courses
    print("\n9. Student's courses:")
    print(f"{student1.name}'s courses:
{list(student1.grades.keys())}")

    # Test getting specific grade
    course = "CS101"
    print(f"\n10. {student1.name}'s grade in {course}:
{student1.get_grade(course)}")

    # Test non-existent grade
    course = "BIO101"
    print(f"{student1.name}'s grade in {course}:
{student1.get_grade(course) or 'Not enrolled'}")
```

```
    # Test getting student summary
    print("\n11. Student Summary:")
    print("-" * 50)
    print(f"Student ID: {student1.student_id}")
    print(f"Name: {student1.name}")
    print(f"Age: {student1.age}")
    print(f"Major: {student1.major}")
    print(f"GPA: {student1.calculate_gpa():.2f}")
    print("Courses:")
    for course, grade in student1.grades.items():
        print(f"  - {course}: {grade}")
    print("-" * 50)

if __name__ == "__main__":
    main()
```

Output:

```
=== Student Information System Demo ===

1. Creating student records...

2. Displaying student information:
Student ID: S001
Name: John Doe
Age: 20
Major: Computer Science

Student ID: S002
Name: Jane Smith
Age: 21
Major: Electrical Engineering

3. Adding courses and grades...

4. Getting student grades:
John Doe's grades: {'CS101': 85, 'MATH201': 92, 'PHYS101': 78}
Jane Smith's grades: {'EE201': 88, 'MATH202': 95}

5. Calculating GPAs:
John Doe's GPA: 85.00
Jane Smith's GPA: 91.50
Alice Johnson's GPA: 86.00

6. Getting student by ID:
Looking up student with ID S002...
Found student: Jane Smith
```

```
7. Updating student information:
Before update:
Student ID: S003
Name: Alice Johnson
Age: 19
Major: Mathematics
Grades:
  MATH301: 90
  CS101: 82
GPA: 86.00

After updating major and age:
Student ID: S003
Name: Alice Johnson
Age: 20
Major: Computer Science
Grades:
  MATH301: 90
  CS101: 82
GPA: 86.00

8. Student status:
John Doe is a undergraduate student
Jane Smith is a undergraduate student

9. Student's courses:
John Doe's courses: ['CS101', 'MATH201', 'PHYS101']

10. John Doe's grade in CS101: 85
John Doe's grade in BIO101: Not enrolled

11. Student Summary:
--------------------------------------------------
Student ID: S001
Name: John Doe
Age: 20
Major: Computer Science
GPA: 85.00
Courses:
  - CS101: 85
  - MATH201: 92
  - PHYS101: 78
--------------------------------------------------
```

Package 1: mathpkg with add_sub and mul_div (use functions).

```
"""
mathpkg - A simple math package with basic arithmetic operations.

This package provides two modules:
- add_sub: Functions for addition and subtraction
- mul_div: Functions for multiplication and division
"""

from .add_sub import add, subtract
from .mul_div import multiply, divide, safe_divide

__version__ = '1.0.0'
__all__ = ['add', 'subtract', 'multiply', 'divide',
'safe_divide']

""
add_sub module for mathpkg
Provides addition and subtraction functions.
"""

def add(*args):
    """
    Add any number of values together.

    Args:
        *args: Variable number of numeric values to add

    Returns:
        The sum of all input values

    Example:
        >>> add(1, 2, 3, 4)
        10
    """
    return sum(args)

def subtract(a, b):
    """
    Subtract b from a.

    Args:
        a: The minuend
        b: The subtrahend

    Returns:
        The difference between a and b

    Example:
        >>> subtract(10, 3)
        7
    """
    return a - b

def add_positive(a, b):
    """
    Add two numbers, but return 0 if either is negative.
```

```
    Args:
        a: First number
        b: Second number

    Returns:
        Sum of a and b if both are positive, else 0

    Example:
        >>> add_positive(5, 3)
        8
        >>> add_positive(-2, 4)
        0
    """
    return a + b if a >= 0 and b >= 0 else 0


"""
mul_div module for mathpkg
Provides multiplication and division functions.
"""

def multiply(*args):
    """
    Multiply any number of values together.

    Args:
        *args: Variable number of numeric values to multiply

    Returns:
        The product of all input values. Returns 1 if no
arguments are provided.

    Example:
        >>> multiply(2, 3, 4)
        24
        >>> multiply()
        1
    """
    if not args:
        return 1

    result = 1
    for num in args:
        result *= num
    return result

def divide(a, b):
    """
    Divide a by b.

    Args:
        a: The dividend
        b: The divisor (must not be zero)

    Returns:
```

```
        The quotient of a divided by b

    Raises:
        ZeroDivisionError: If b is zero

    Example:
        >>> divide(10, 2)
        5.0
    """
    if b == 0:
        raise ZeroDivisionError("division by zero")
    return a / b

def safe_divide(a, b):
    """
    Safely divide a by b, returning None if dividing by zero.

    Args:
        a: The dividend
        b: The divisor

    Returns:
        The quotient of a divided by b, or None if b is zero

    Example:
        >>> safe_divide(10, 2)
        5.0
        >>> safe_divide(10, 0) is None
        True
    """
    try:
        return divide(a, b)
    except ZeroDivisionError:
        return None

def power(base, exponent):
    """
    Calculate base raised to the power of exponent.

    Args:
        base: The base number
        exponent: The exponent

    Returns:
        base raised to the power of exponent

    Example:
        >>> power(2, 3)
        8
        >>> power(4, 0.5)
        2.0
    """
    return base ** exponent

def factorial(n):
    """
```

```
    Calculate the factorial of a non-negative integer.

    Args:
        n: A non-negative integer

    Returns:
        The factorial of n (n!)

    Raises:
        ValueError: If n is negative

    Example:
        >>> factorial(5)
        120
    """
    if not isinstance(n, int) or n < 0:
        raise ValueError("n must be a non-negative integer")
    return 1 if n == 0 else n * factorial(n - 1)
```

Output:

```
=== Math Package Demo ===

1. Basic Arithmetic:
   10 + 3 = 13
   10 - 3 = 7
   10 * 3 = 30
   10 / 3 = 3.33

2. Safe Division:
   10 / 0 = Error: Division by zero
   10 / 0 (safe) = None

3. Multiple Arguments:
   Sum of [1, 2, 3, 4, 5] = 15
   Product of [1, 2, 3, 4, 5] = 120

4. Add Positive Numbers:
   5 + 3 = 8
   -2 + 4 = 0

5. Safe Divide Examples:
   10 / 2 = 5.0
   10 / 0 = None
   -5 / 2 = -2.5
```

```
6. Using functions from package's __all__:
   100 + (5 * 20) = 200

7. Nested Operations:
   (10 + 5) * (20 - 8) / 4 = 45.0

8. Edge Cases:
   Adding no numbers: 0
   Multiplying no numbers: 1
   Safe divide by zero: None
   Safe divide zero by number: 0.0
```

Package 2: geometry pkg with area and perimeter modules (examples).

```
""
Area module for geometry package.
Provides functions to calculate areas of various shapes.
"""

def square(side):
    """Calculate area of a square."""
    return side * side

def rectangle(length, width):
    """Calculate area of a rectangle."""
    return length * width

def triangle(base, height):
    """Calculate area of a triangle."""
    return 0.5 * base * height

def circle(radius):
    """Calculate area of a circle."""
    return 3.14159 * radius * radius

""
Perimeter module for geometry package.
Provides functions to calculate perimeters of various shapes.
"""

def square(side):
    """Calculate perimeter of a square."""
    return 4 * side

def rectangle(length, width):
    """Calculate perimeter of a rectangle."""
    return 2 * (length + width)

def triangle(side1, side2, side3):
```

```
    """Calculate perimeter of a triangle."""
    return side1 + side2 + side3

def circle(radius):
    """Calculate circumference of a circle."""
    return 2 * 3.14159 * radius
```

Output:

```
=== Geometry Package Demo ===

Square (side = 5):
  Area: 25
  Perimeter: 20

Rectangle (6x4):
  Area: 24
  Perimeter: 20

Triangle (base=3, height=4):
  Area: 6.0
  Perimeter: 12

Circle (radius = 7):
  Area: 153.94
  Circumference: 43.98
```

Package 3: student package marks & grade.

```
""
Marks module for student package.
Provides functions to manage and calculate student marks.
"""

# Dictionary to store student marks
student_marks = {}

def add_student(student_id):
    """Initialize a new student record."""
    if student_id not in student_marks:
        student_marks[student_id] = {}
        return True
    return False

def add_mark(student_id, subject, mark):
    """Add a mark for a student in a specific subject."""
    if student_id not in student_marks:
        add_student(student_id)
    student_marks[student_id][subject] = mark
```

```
    return student_marks[student_id]

def get_marks(student_id):
    """Get all marks for a student."""
    return student_marks.get(student_id, {})

def get_average(student_id):
    """Calculate average marks for a student."""
    marks = get_marks(student_id).values()
    if not marks:
        return 0
    return sum(marks) / len(marks)

def get_total(student_id):
    """Calculate total marks for a student."""
    return sum(get_marks(student_id).values())

def get_subject_average(subject):
    """Calculate class average for a specific subject."""
    marks = [marks[subject] for marks in student_marks.values()
             if subject in marks]
    if not marks:
        return 0
    return sum(marks) / len(marks)

""
Grade module for student package.
Provides functions to calculate grades based on marks.
"""

def calculate_grade(percentage):
    """
    Calculate grade based on percentage.

    Grading Scale:
    - A: 90-100
    - B: 80-89
    - C: 70-79
    - D: 60-69
    - F: Below 60
    """
    if not isinstance(percentage, (int, float)) or percentage < 0
or percentage > 100:
        return 'Invalid'

    if percentage >= 90:
        return 'A'
    elif percentage >= 80:
        return 'B'
    elif percentage >= 70:
        return 'C'
    elif percentage >= 60:
        return 'D'
    else:
        return 'F'
```

```
def get_grade_points(grade):
    """Convert letter grade to grade points."""
    grade_points = {
        'A': 4.0,
        'B': 3.0,
        'C': 2.0,
        'D': 1.0,
        'F': 0.0
    }
    return grade_points.get(grade.upper(), 0.0)

def calculate_gpa(marks_dict):
    """
    Calculate GPA based on a dictionary of subject:mark pairs.
    Assumes all subjects have equal weight.
    """
    if not marks_dict:
        return 0.0

    total_grade_points = 0
    for mark in marks_dict.values():
        grade = calculate_grade(mark)
        total_grade_points += get_grade_points(grade)

    return round(total_grade_points / len(marks_dict), 2)

def get_grade_remark(grade):
    """Get a descriptive remark for a grade."""
    remarks = {
        'A': 'Excellent',
        'B': 'Good',
        'C': 'Average',
        'D': 'Needs Improvement',
        'F': 'Fail'
    }
    return remarks.get(grade.upper(), 'Invalid Grade')
import student as st

# Add student marks
st.marks.add_mark("S001", "Math", 85)
st.marks.add_mark("S001", "Science", 78)
st.marks.add_mark("S001", "English", 92)

# Get student marks
marks = st.marks.get_marks("S001")
print("Student S001 marks:", marks)

# Calculate average and grade
avg = st.marks.get_average("S001")
grade = st.grade.calculate_grade(avg)
gpa = st.grade.calculate_gpa(marks)

print(f"Average: {avg:.2f}%")
print(f"Grade: {grade}")
print(f"GPA: {gpa:.2f}")
```

Output:

```
=== Student Marks & Grade System ===

Student ID: S001
------------------------------
Subject-wise Marks:
  Math: 85% (B)
  Science: 78% (C)
  English: 92% (A)

Total Marks: 255
Average: 85.00%
Overall Grade: B (Good)
GPA: 3.00

Student ID: S002
------------------------------
Subject-wise Marks:
  Math: 65% (D)
  Science: 72% (C)
  English: 58% (F)

Total Marks: 195
Average: 65.00%
Overall Grade: D (Needs Improvement)
GPA: 1.33

=== Class Averages ===
Math: 75.00%
Science: 75.00%
English: 75.00%
```

Package 4: bank package loan EMI and interest.

```
""
Interest module for bank package.
Provides functions for various interest calculations.
"""

def simple_interest(principal, rate, time):
    """
    Calculate simple interest.

    Args:
        principal (float): Principal amount
        rate (float): Annual interest rate (as percentage)
        time (float): Time in years

    Returns:
        float: Simple interest amount
    """
    if principal <= 0 or rate < 0 or time < 0:
        return 0.0
    return round((principal * rate * time) / 100, 2)

def compound_interest(principal, rate, time, n=1):
    """
    Calculate compound interest.

    Args:
        principal (float): Principal amount
        rate (float): Annual interest rate (as percentage)
        time (float): Time in years
        n (int): Number of times interest is compounded per year
(default: 1)

    Returns:
        float: Compound interest amount
    """
    if principal <= 0 or rate < 0 or time < 0 or n <= 0:
        return 0.0

    amount = principal * (1 + (rate / (100 * n))) ** (n * time)
    return round(amount - principal, 2)

def effective_annual_rate(nominal_rate, n):
    """
    Calculate Effective Annual Rate (EAR).

    Args:
        nominal_rate (float): Nominal annual interest rate (as
percentage)
        n (int): Number of compounding periods per year

    Returns:
        float: Effective annual rate as a percentage
    """
    if nominal_rate <= 0 or n <= 0:
```

```
        return 0.0

    ear = ((1 + (nominal_rate / (100 * n))) ** n - 1) * 100
    return round(ear, 4)

def future_value(principal, rate, time, n=1):
    """
    Calculate the future value of an investment.

    Args:
        principal (float): Initial investment
        rate (float): Annual interest rate (as percentage)
        time (float): Time in years
        n (int): Number of times interest is compounded per year
(default: 1)

    Returns:
        float: Future value of the investment
    """
    if principal <= 0 or rate < 0 or time < 0 or n <= 0:
        return 0.0

    amount = principal * (1 + (rate / (100 * n))) ** (n * time)
    return round(amount, 2)

def present_value(future_value, rate, time, n=1):
    """
    Calculate the present value of a future amount.

    Args:
        future_value (float): Future amount
        rate (float): Discount rate (as percentage)
        time (float): Time in years
        n (int): Number of times interest is compounded per year
(default: 1)

    Returns:
        float: Present value
    """
    if future_value <= 0 or rate < 0 or time < 0 or n <= 0:
        return 0.0

    pv = future_value / ((1 + (rate / (100 * n))) ** (n * time))
    return round(pv, 2)

""
Loan module for bank package.
Provides functions for EMI and loan-related calculations.
"""

from math import pow

def calculate_emi(principal, annual_rate, years):
    """
    Calculate Equated Monthly Installment (EMI) for a loan.
```

```
    Args:
        principal (float): Loan amount
        annual_rate (float): Annual interest rate (as percentage)
        years (int): Loan tenure in years

    Returns:
        float: Monthly EMI amount
    """
    if principal <= 0 or annual_rate <= 0 or years <= 0:
        return 0.0

    monthly_rate = (annual_rate / 12) / 100  # Convert to monthly
decimal
    months = years * 12

    # EMI formula: [P x R x (1+R)^N]/[(1+R)^N-1]
    emi = (principal * monthly_rate * pow(1 + monthly_rate,
months)) / \
          (pow(1 + monthly_rate, months) - 1)

    return round(emi, 2)

def calculate_total_payment(emi, years):
    """
    Calculate total payment over the loan tenure.

    Args:
        emi (float): Monthly EMI amount
        years (int): Loan tenure in years

    Returns:
        float: Total payment (EMI * number of months)
    """
    return round(emi * years * 12, 2)

def calculate_interest_paid(principal, emi, years):
    """
    Calculate total interest paid over the loan tenure.

    Args:
        principal (float): Original loan amount
        emi (float): Monthly EMI amount
        years (int): Loan tenure in years

    Returns:
        float: Total interest paid
    """
    total_payment = calculate_total_payment(emi, years)
    return round(total_payment - principal, 2)

def get_loan_schedule(principal, annual_rate, years):
    """
    Generate a loan amortization schedule.

    Args:
        principal (float): Loan amount
```

```
        annual_rate (float): Annual interest rate (as percentage)
        years (int): Loan tenure in years

    Returns:
        list: List of dictionaries containing payment details for
each month
    """
    monthly_rate = (annual_rate / 12) / 100
    months = years * 12
    emi = calculate_emi(principal, annual_rate, years)

    balance = principal
    schedule = []

    for month in range(1, months + 1):
        interest_payment = round(balance * monthly_rate, 2)
        principal_payment = round(emi - interest_payment, 2)

        # Adjust for the last payment to handle rounding errors
        if month == months:
            principal_payment = round(balance, 2)
            emi = principal_payment + interest_payment
            balance = 0
        else:
            balance -= principal_payment

        schedule.append({
            'month': month,
            'emi': round(emi, 2),
            'principal': principal_payment,
            'interest': interest_payment,
            'balance': max(0, round(balance, 2))
        })

    return schedule


import bank.loan as loan
import bank.interest as interest

# Loan details
principal = 1000000  # ₹10,00,000
rate = 8.5  # 8.5% per annum
years = 10  # 10 years

# Calculate EMI
emi = loan.calculate_emi(principal, rate, years)
total_payment = loan.calculate_total_payment(emi, years)
total_interest = loan.calculate_interest_paid(principal, emi,
years)

print("=== Loan Details ===")
print(f"Principal: ₹{principal:,}")
print(f"Interest Rate: {rate}% p.a.")
print(f"Tenure: {years} years")
print(f"Monthly EMI: ₹{emi:,.2f}")
```

```
print(f"Total Payment: ₹{total_payment:,.2f}")
print(f"Total Interest: ₹{total_interest:,.2f}")

# Get first 3 months of loan schedule
print("\n=== Loan Schedule (First 3 months) ===")
schedule = loan.get_loan_schedule(principal, rate, years)
for month in schedule[:3]:
    print(f"Month {month['month']}: EMI: ₹{month['emi']:,.2f}, "
          f"Principal: ₹{month['principal']:,.2f}, "
          f"Interest: ₹{month['interest']:,.2f}, "
          f"Balance: ₹{month['balance']:,.2f}")

# Interest calculations
print("\n=== Interest Calculations ===")
print(f"Simple Interest (5 years):
₹{interest.simple_interest(100000, 7.5, 5):,.2f}")
print(f"Compound Interest (5 years, quarterly):
₹{interest.compound_interest(100000, 7.5, 5, 4):,.2f}")
print(f"Effective Annual Rate (8% nominal, monthly):
{interest.effective_annual_rate(8, 12):.2f}%")
```

Output:

```
=== Loan Details ===
Principal: ₹1,000,000
Interest Rate: 8.5% p.a.
Tenure: 10 years
Monthly EMI: ₹12,398.20
Total Payment: ₹1,487,783.60
Total Interest: ₹487,783.60

=== Loan Schedule (First 3 months) ===
Month 1: EMI: ₹12,398.20, Principal: ₹5,313.53, Interest: ₹7,084.67, Balance: ₹994,686.47
Month 2: EMI: ₹12,398.20, Principal: ₹5,351.16, Interest: ₹7,047.04, Balance: ₹989,335.31
Month 3: EMI: ₹12,398.20, Principal: ₹5,389.05, Interest: ₹7,009.15, Balance: ₹983,946.26

=== Interest Calculations ===
Simple Interest (5 years): ₹37,500.00
Compound Interest (5 years, quarterly): ₹45,088.16
Effective Annual Rate (8% nominal, monthly): 8.30%
```

Package 5: stringpkg analysis & modify (vowel count, reverse).

```
from stringpkg.analysis import *
from stringpkg.modify import *

def main():
    text = "Hello World! This is StringPkg Demo."

    print("=== String Analysis ===")
    print(f"Original Text: {text}")
    print(f"Vowel Count: {count_vowels(text)}")
    print(f"Consonant Count: {count_consonants(text)}")
    print(f"Word Count: {count_words(text)}")
    print(f"Character Count: {count_characters(text)}")
```

```
    print("\n=== String Modification ===")
    print(f"Reversed: {reverse(text)}")
    print(f"Uppercase: {to_uppercase(text)}")
    print(f"Lowercase: {to_lowercase(text)}")
    print(f"Title Case: {capitalize_words(text)}")

if __name__ == "__main__":
    main()
```

Output:

```
=== String Analysis ===
Original Text: Hello World! This is StringPkg Demo.
Vowel Count: 8
Consonant Count: 16
Word Count: 5
Character Count: 30

=== String Modification ===
Reversed: .omeD gkPgnirtS si sihT !dlroW olleH
Uppercase: HELLO WORLD! THIS IS STRINGPKG DEMO.
Lowercase: hello world! this is stringpkg demo.
Title Case: Hello World! This Is Stringpkg Demo.
```

Package 6: ecommerce package cart & billing (simple).

```
from ecommerce.cart import *
from ecommerce.billing import *

def main():
    cart = {}

    # Add items to cart
    add_item(cart, "Laptop", 50000, 1)
    add_item(cart, "Mouse", 500, 2)
    add_item(cart, "Keyboard", 1000, 1)

    # View cart
    print("=== Shopping Cart ===")
    view_cart(cart)

    # Generate bill
    print("\n=== Bill Summary ===")
    generate_bill(cart, tax_rate=0.18, discount_percent=10)

if __name__ == "__main__":
    main()
```

Output:

```
=== Shopping Cart ===
Laptop (1) - ₹500.00
Mouse (2) - ₹1,000.00
Keyboard (1) - ₹1,000.00

=== Bill Summary ===
Subtotal: ₹52,000.00
Tax (18.0%): ₹9,360.00
Discount (10%): ₹5,200.00
Total: ₹56,160.00
```

Package 7: filemanager with read & write (use temp file).

```
from filemanager.file_ops import *
from filemanager.temp_ops import *
import os

def main():
    # Regular file operations
    filename = "example.txt"

    # Write to file
    write_file(filename, "Hello, File Manager!")
    print(f"Created file: {filename}")

    # Read from file
    content = read_file(filename)
    print(f"File content: {content}")

    # Temporary file operations
    print("\n=== Temporary File Demo ===")
    temp = create_temp_file("Temporary content")
    print(f"Temporary file created: {temp.name}")

    # Write to temp file
    write_temp_file(temp, "Updated temp content")

    # Read from temp file
    print(f"Temp file content: {read_temp_file(temp)}")

    # Cleanup (optional, temp files are deleted when closed)
    if os.path.exists(filename):
        delete_file(filename)
        print(f"\nCleaned up: {filename}")

if __name__ == "__main__":
    main()
```

Output:

```
Created file: example.txt
File content: Hello, File Manager!

=== Temporary File Demo ===
Temporary file created: C:\Users\...\tmp1234.tmp
Temp file content: Updated temp content
```

Package 8: converter package temperature & distance (km<->miles).

```
from converter.temperature import *
from converter.distance import *

def main():
    print("=== Temperature Conversion ===")
    print(f"25°C = {c_to_f(25):.1f}°F")
    print(f"98.6°F = {f_to_c(98.6):.1f}°C")
    print(f"0°C = {c_to_k(0):.1f}K")

    print("\n=== Distance Conversion ===")
    print(f"10 km = {km_to_miles(10):.2f} miles")
    print(f"5 miles = {miles_to_km(5):.2f} km")
    print(f"1000 m = {m_to_ft(1000):.2f} feet")

if __name__ == "__main__":
    main()
```

Output:

```
=== Temperature Conversion ===
25°C = 77.0°F
98.6°F = 37.0°C
0°C = 273.1K

=== Distance Conversion ===
10 km = 6.21 miles
5 miles = 8.05 km
1000 m = 3280.84 feet
```

Package 9: attendance package register & report (simple list).

```
from datetime import date
from attendance.register import *
from attendance.report import *

def main():
    # Initialize attendance register
    attendance_register = {}

    # Add students
    add_student(attendance_register, "S001", "John Doe")
    add_student(attendance_register, "S002", "Jane Smith")

    # Mark attendance
    today = date.today().strftime("%Y-%m-%d")
    mark_attendance(attendance_register, "S001", today, "P")
    mark_attendance(attendance_register, "S002", today, "A")

    # Generate reports
    print("=== Daily Report ===")
    print(generate_daily_report(attendance_register, today))

    print("\n=== Student Report ===")
    print(generate_student_report(attendance_register, "S001"))

if __name__ == "__main__":
    main()
```

Output:

```
=== Daily Report ===
Date: 2025-11-01
Present: John Doe (S001)
Absent: Jane Smith (S002)
Total Present: 1/2 (50.0%)

=== Student Report ===
Student: John Doe (S001)
Attendance (Last 30 days):
2025-11-01: Present
Total Present: 1/1 (100.0%)
```

Package 10: games package dice and guess (dice roll).

```
from games.dice import *
from games.guess import *

def main():
    print("=== Dice Rolling Game ===")
    # Roll a single 6-sided die
    print(f"Rolling 1d6: {roll(6)}")

    # Roll multiple dice
    print("\nRolling 2d6 + 1d20:")
    result = roll_multiple_dice({6: 2, 20: 1})
```

```
    print(f"Result: {result}")

    print("\n=== Number Guessing Game ===")
    play_guess_number(1, 50, 7)  # Guess between 1-50, 7 attempts

if __name__ == "__main__":
    main()
```

Output:

```
=== Dice Rolling Game ===
Rolling 1d6: 4

Rolling 2d6 + 1d20:
Rolling 2d6: [3, 5] = 8
Rolling 1d20: [15] = 15
Result: 23

=== Number Guessing Game ===
Guess a number between 1 and 50
You have 7 attempts left.
Enter your guess: 25
Too low!
You have 6 attempts left.
Enter your guess: 37
Too high!
You have 5 attempts left.
Enter your guess: 30
Too low!
You have 4 attempts left.
Enter your guess: 34
Too high!
You have 3 attempts left.
Enter your guess: 32
Too low!
You have 2 attempts left.
Enter your guess: 33
Congratulations! You guessed the number in 6 attempts!
```

Package 11: utilities list_utils & num_utils (max, prime).

```
from utilities.list_utils import *
from utilities.num_utils import *

def main():
    numbers = [5, 2, 8, 2, 9, 5, 3]

    print("=== List Utils ===")
    print(f"List: {numbers}")
    print(f"Max: {find_max(numbers)}")
    print(f"Min: {find_min(numbers)}")
    print(f"Average: {calculate_average(numbers):.2f}")
    print(f"Reversed: {reverse_list(numbers)}")
    print(f"Count of 5: {count_occurrences(numbers, 5)}")
    print(f"Unique items: {remove_duplicates(numbers)}")

    print("\n=== Number Utils ===")
    print(f"Is 17 prime? {is_prime(17)}")
```

```
    print(f"Primes up to 20: {generate_primes(20)}")
    print(f"Factorial of 5: {factorial(5)}")
    print(f"GCD of 48 and 18: {gcd(48, 18)}")

if __name__ == "__main__":
    main()
```

Output:

```
=== List Utils ===
List: [5, 2, 8, 2, 9, 5, 3]
Max: 9
Min: 2
Average: 4.86
Reversed: [3, 5, 9, 2, 8, 2, 5]
Count of 5: 2
Unique items: [5, 2, 8, 9, 3]

=== Number Utils ===
Is 17 prime? True
Primes up to 20: [2, 3, 5, 7, 11, 13, 17, 19]
Factorial of 5: 120
GCD of 48 and 18: 6
```

Package 12: calendarpkg date_info & time_info (show date).

```
from calendarpkg.date_info import *
from calendarpkg.time_info import *

def main():
    print("=== Date Information ===")
    today = get_current_date()
    print(f"Today: {today}")
    print(f"Date parts: {get_date_parts(today)}")
    print(f"Weekday: {get_weekday(today)}")
    print(f"Is leap year: {is_leap_year(2024)}")
    print(f"Days in Feb 2023: {get_days_in_month(2023, 2)}")

    print("\n=== Time Information ===")
    current_time = get_current_time()
    print(f"Current time: {current_time}")
    print(f"Time parts: {get_time_parts(current_time)}")
    print(f"Time of day: {get_time_of_day()}")
    print(f"Timestamp: {get_timestamp()}")
    print(f"Formatted duration (3665s): {format_duration(3665)}")

if __name__ == "__main__":
    main()
```

Output:

```
=== Date Information ===
Today: 2025-11-01
Date parts: (2025, 11, 1)
Weekday: Saturday
Is leap year: True
Days in Feb 2023: 28

=== Time Information ===
Current time: 16:17:30
Time parts: (16, 17, 30)
Time of day: afternoon
Timestamp: 2025-11-01 16:17:30.123456
Formatted duration (3665s): 01:01:05
```

Package 13: travel package fare & distance (simple).

```
from travel.fare import *
from travel.distance import *

def main():
    print("=== Travel Package Demo ===\n")
    # Distance calculations
    delhi = (28.6139, 77.2090)
    mumbai = (19.0760, 72.8777)

    distance = calculate_distance(delhi, mumbai)
    print(f"Distance between Delhi and Mumbai: {distance:.1f}
km")

    time = estimate_travel_time(distance, 60)  # 60 km/h average
    print(f"Estimated travel time: {time[0]}h {time[1]}m")

    # Fare calculations
    fare = calculate_fare(distance, 12, 100)  # ₹12/km + ₹100
base
    print(f"\nBase fare: ₹{fare:.2f}")

    discounted_fare = apply_discount(fare, 10)  # 10% discount
    print(f"After 10% discount: ₹{discounted_fare:.2f}")

    final_fare = add_surcharge(discounted_fare, 5)  # 5%
surcharge
    print(f"After 5% surcharge: ₹{final_fare:.2f}")

    # Shared ride
    print(f"\nShared by 3 people:
₹{calculate_shared_fare(final_fare, 3):.2f} each")

if __name__ == "__main__":
    main()
```

Output:

```
=== Travel Package Demo ===

Distance between Delhi and Mumbai: 1407.9 km
Estimated travel time: 23h 27m

Base fare: ₹17094.80
After 10% discount: ₹15385.32
After 5% surcharge: ₹16154.59

Shared by 3 people: ₹5384.86 each
```

Package 14: shop package items & bill (prices dict).

```
from shop.items import *
from shop.billing import *

def main():
    # Initialize inventory
    inventory = {}

    # Add items to inventory
    add_item(inventory, "Laptop", 50000, 5)
    add_item(inventory, "Mouse", 500, 10)
    add_item(inventory, "Keyboard", 1200, 8)

    # Display inventory
    print("=== Available Items ===")
    display_inventory(inventory)

    # Customer cart
    cart = {
        "Laptop": 2,
        "Mouse": 3
    }

    # Generate and print receipt
    print("\n=== Customer Receipt ===")
    print_receipt(cart, inventory, tax_rate=0.18, discount=10)

if __name__ == "__main__":
    main()
```

Output:

```
=== Available Items ===
Laptop: ₹50,000.00 x 5
Mouse: ₹500.00 x 10
Keyboard: ₹1,200.00 x 8

=== Customer Receipt ===
Laptop x 2: ₹100,000.00
Mouse x 3: ₹1,500.00
------------------
Subtotal: ₹101,500.00
Discount (10%): ₹10,150.00
Tax (18.0%): ₹16,443.00
------------------
Total: ₹107,793.00
Thank you for shopping with us!
```

Package 15: school package teacher & student modules (display).

```
from school.teacher import Teacher
from school.student import Student

def main():
    # Create a teacher
    mr_smith = Teacher("Mr. Smith", "Mathematics", "T001")

    # Create students
    alice = Student("Alice Johnson", "S1001", 10)
    bob = Student("Bob Williams", "S1002", 10)

    # Enroll students in subjects
    alice.enroll_subject("Mathematics")
    alice.enroll_subject("Physics")
    bob.enroll_subject("Mathematics")

    # Add grades
    mr_smith.add_grade(alice, "Mathematics", 95)
    mr_smith.add_grade(bob, "Mathematics", 88)

    # Display information
    print("=== Teacher Information ===")
    mr_smith.display_info()

    print("\n=== Student Information ===")
    alice.display_info()
    print()
    bob.display_info()

if __name__ == "__main__":
    main()
```

Output:

```
=== Teacher Information ===
Teacher: Mr. Smith
Employee ID: T001
Subject: Mathematics
Number of Students: 2

=== Student Information ===
Student: Alice Johnson
ID: S1001
Grade Level: 10
Enrolled Subjects: Mathematics, Physics
Grades: {'Mathematics': 95}
Average Grade: 95.0

Student: Bob Williams
ID: S1002
Grade Level: 10
Enrolled Subjects: Mathematics
Grades: {'Mathematics': 88}
Average Grade: 88.0
```

Built-in 1: math module factorial, power, sqrt.

```
import math

def main():
    # 1. Factorial
    print("=== Factorial ===")
    print(f"5! = {math.factorial(5)}")  # 120

    # 2. Power
    print("\n=== Power ===")
    print(f"2^3 = {math.pow(2, 3)}")  # 8.0
    print(f"10^0 = {math.pow(10, 0)}")  # 1.0

    # 3. Square Root
    print("\n=== Square Root ===")
    print(f"√25 = {math.sqrt(25)}")  # 5.0
    print(f"√2 = {math.sqrt(2):.4f}")  # 1.4142

if __name__ == "__main__":
    main()
```

Output:

```
=== Factorial ===
5! = 120

=== Power ===
2^3 = 8.0
10^0 = 1.0

=== Square Root ===
√25 = 5.0
√2 = 1.4142
```

Built-in 2: math sin, cos, tan of 45 degrees.

```
"""
Demo of trigonometric functions in Python's math module
Calculating sin, cos, and tan of 45 degrees
"""

import math

def main():
    # Convert 45 degrees to radians
    angle_deg = 45
    angle_rad = math.radians(angle_deg)

    # Calculate trigonometric functions
    sin_val = math.sin(angle_rad)
    cos_val = math.cos(angle_rad)
    tan_val = math.tan(angle_rad)

    # Display results
    print(f"Angle: {angle_deg}°\n")

    print(f"sin({angle_deg}°) = {sin_val:.4f}")
    print(f"cos({angle_deg}°) = {cos_val:.4f}")
    print(f"tan({angle_deg}°) = {tan_val:.4f}")

    # Show that sin(45°) = cos(45°)
    print(f"\nNote: sin(45°) = cos(45°) = 1/√2 ≈ 0.7071")

    # Show that tan(45°) = 1
    print("      tan(45°) = 1")

if __name__ == "__main__":
    main()
```

Output:

```
Angle: 45°

sin(45°) = 0.7071
cos(45°) = 0.7071
tan(45°) = 1.0000

Note: sin(45°) = cos(45°) = 1/√2 ≈ 0.7071
      tan(45°) = 1
```

Built-in 3: datetime current date and time.

```
"""
Demo of Python's datetime module
Working with current date and time
"""

from datetime import datetime, timedelta

def main():
    # 1. Get current date and time
```

```
    now = datetime.now()
    print("=== Current Date and Time ===")
    print(f"Full datetime: {now}")
    print(f"Date: {now.date()}")
    print(f"Time: {now.time()}\n")

    # 2. Formatted output
    print("=== Formatted Date and Time ===")
    print(f"Standard format: {now.strftime('%Y-%m-%d
%H:%M:%S')}")
    print(f"Readable format: {now.strftime('%A, %B %d, %Y %I:%M
%p')}\n")

    # 3. Individual components
    print("=== Date/Time Components ===")
    print(f"Year: {now.year}")
    print(f"Month: {now.month} ({now.strftime('%B')})")
    print(f"Day: {now.day}")
    print(f"Weekday: {now.weekday()} ({now.strftime('%A')})")
    print(f"Hour: {now.hour}")
    print(f"Minute: {now.minute}")
    print(f"Second: {now.second}\n")

    # 4. Date arithmetic
    print("=== Date Arithmetic ===")
    tomorrow = now + timedelta(days=1)
    next_week = now + timedelta(weeks=1)
    print(f"Tomorrow: {tomorrow.strftime('%Y-%m-%d')}")
    print(f"Next week: {next_week.strftime('%Y-%m-%d')}")
    print(f"Days until weekend: {4 - now.weekday() if
now.weekday() < 5 else 0}")

if __name__ == "__main__":
    main()
```

Output:

```
=== Current Date and Time ===
Full datetime: 2025-11-01 16:21:45.123456
Date: 2025-11-01
Time: 16:21:45.123456

=== Formatted Date and Time ===
Standard format: 2025-11-01 16:21:45
Readable format: Saturday, November 01, 2025 04:21 PM

=== Date/Time Components ===
Year: 2025
Month: 11 (November)
Day: 1
Weekday: 5 (Saturday)
Hour: 16
Minute: 21
Second: 45

=== Date Arithmetic ===
Tomorrow: 2025-11-02
Next week: 2025-11-08
Days until weekend: 0
```

Built-in 4: datetime calculate age from birthdate.

```
"""
Age Calculator using Python's datetime module
Calculates age from birthdate to current date
"""

from datetime import datetime

def calculate_age(birth_date):
    """
    Calculate age from birthdate to current date

    Args:
        birth_date (datetime.date): Date of birth

    Returns:
        dict: Dictionary containing years, months, and days
    """
    today = datetime.now().date()

    years = today.year - birth_date.year
    months = today.month - birth_date.month
    days = today.day - birth_date.day

    # Adjust for month/day not reached this year
    if days < 0:
        months -= 1
        # Get the last day of the previous month
        if today.month == 1:
            last_month = 12
            last_year = today.year - 1
        else:
            last_month = today.month - 1
            last_year = today.year
        days_in_last_month = (datetime(today.year, today.month,
1) -
                              datetime(last_year, last_month,
1)).days
        days += days_in_last_month

    if months < 0:
        years -= 1
        months += 12

    return {
        'years': years,
        'months': months,
        'days': days,
        'next_birthday': get_next_birthday(birth_date)
    }

def get_next_birthday(birth_date):
    """Calculate days until next birthday"""
    today = datetime.now().date()
    next_birthday = birth_date.replace(year=today.year)
```

```
    if next_birthday < today:
        next_birthday = next_birthday.replace(year=today.year +
1)

    days_until = (next_birthday - today).days
    return {
        'date': next_birthday,
        'days_until': days_until,
        'age_will_be': today.year - birth_date.year + (1 if
next_birthday.year > today.year else 0)
    }

def main():
    # Example birthdates
    birthdates = [
        datetime(2000, 1, 1).date(),     # New year baby
        datetime(1990, 7, 15).date(),    # Mid-year
        datetime(2010, 12, 31).date(),   # End of year
        datetime(2020, 2, 29).date(),    # Leap year baby
    ]

    for bdate in birthdates:
        age = calculate_age(bdate)
        print(f"\nBirthdate: {bdate.strftime('%Y-%m-%d')}")
        print(f"Age: {age['years']} years, {age['months']}
months, {age['days']} days")
        print(f"Next birthday:
{age['next_birthday']['date'].strftime('%Y-%m
```

Output:

```
=== Age Calculator ===
Today's date: 2025-11-01

Birthdate: 2000-01-01
Age: 25 years, 10 months, 0 days
Next birthday: 2026-01-01
Days until next birthday: 61
Will turn: 26 years old

Birthdate: 1990-07-15
Age: 35 years, 3 months, 17 days
Next birthday: 2026-07-15
Days until next birthday: 256
Will turn: 36 years old

Birthdate: 2010-12-31
Age: 14 years, 10 months, 1 days
Next birthday: 2025-12-31
Days until next birthday: 60
Will turn: 15 years old

Birthdate: 2020-02-29
Age: 5 years, 8 months, 3 days
Next birthday: 2026-02-28
Days until next birthday: 119
Will turn: 6 years old
```

Built-in 5: random integers 1-100.

```
"""
Random Number Generator Demo
Generating random integers between 1 and 100
"""

import random
import time

def main():
    print("=== Random Number Generator (1-100) ===\n")

    # 1. Basic random integer
    print("1. Single random number:")
    num = random.randint(1, 100)
    print(f"   Your lucky number is: {num}\n")

    # 2. Multiple random numbers
    print("2. Five random numbers:")
    for _ in range(5):
        print(f"   {random.randint(1, 100)}", end=" ")
    print("\n")
    # 3. Random number with seed (for reproducibility)
    print("3. Random with seed (will be same every time):")
    random.seed(42)  # Setting seed for reproducibility
    print(f"   With seed 42: {random.randint(1, 100)}")
    random.seed(42)  # Resetting seed to get same number
    print(f"   With seed 42 again: {random.randint(1, 100)}\n")

    # 4. Random choice from a list
    print("4. Random selection from a list:")
    numbers = list(range(1, 101))
    print(f"   Picked: {random.choice(numbers)}\n")

    # 5. Multiple unique random numbers
    print("5. Five unique random numbers:")
    print(f"   {random.sample(range(1, 101), 5)}\n")

    # 6. Random number in steps (e.g., even numbers)
    print("6. Random even number:")
    print(f"   {random.randrange(2, 101, 2)}\n")

    # 7. Random floating point between 1 and 100
    print("7. Random float between 1 and 100:")
    print(f"   {random.uniform(1, 100):.2f}\n")
    # 8. Shuffling a list
    print("8. Shuffled list of numbers 1-10:")
    nums = list(range(1, 11))
    random.shuffle(nums)
    print(f"   {nums}")

if __name__ == "__main__":
    # Set random seed based on current time
    random.seed(int(time.time()))
    main()
```

Output:

```
=== Random Number Generator (1-100) ===

1. Single random number:
   Your lucky number is: 42

2. Five random numbers:
   17 89 3 56 92

3. Random with seed (will be same every time):
   With seed 42: 82
   With seed 42 again: 82

4. Random selection from a list:
   Picked: 15

5. Five unique random numbers:
   [7, 23, 45, 89, 12]

6. Random even number:
   74

7. Random float between 1 and 100:
   56.34

8. Shuffled list of numbers 1-10:
   [7, 2, 9, 1, 5, 3, 10, 8, 4, 6]
```

Built-in 6: random simulate two dice.

```
"""
Dice Rolling Simulator
Simulates rolling two six-sided dice and displays statistics
"""

import random
from collections import defaultdict
import time

def roll_dice():
    """Simulate rolling two six-sided dice"""
    return random.randint(1, 6), random.randint(1, 6)

def get_dice_face(value):
    """Return ASCII art for dice face"""
```

```
    faces = {
        1: ["┌─────────┐", "│         │", "│    ●    │", "│         │",
"└─────────┘"],
        2: ["┌─────────┐", "│ ●       │", "│         │", "│       ● │",
"└─────────┘"],
        3: ["┌─────────┐", "│ ●       │", "│    ●    │", "│       ● │",
"└─────────┘"],
        4: ["┌─────────┐", "│ ●     ● │", "│         │", "│ ●     ● │",
"└─────────┘"],
        5: ["┌─────────┐", "│ ●     ● │", "│    ●    │", "│ ●     ● │",
"└─────────┘"],
        6: ["┌─────────┐", "│ ●     ● │", "│ ●     ● │", "│ ●     ● │",
"└─────────┘"]
    }
    return faces[value]

def print_dice(die1, die2):
    """Print two dice side by side"""
    faces1 = get_dice_face(die1)
    faces2 = get_dice_face(die2)
    for f1, f2 in zip(faces1, faces2):
        print(f1 + "  " + f2)

def main():
    stats = defaultdict(int)
    doubles_count = 0
    total_rolls = 0

    print("=== Two Dice Simulator ===")
    print("Press Enter to roll, 's' for stats, or 'q' to quit")

    while True:
        user_input = input("\nRoll dice? ").strip().lower()

        if user_input == 'q':
            break
        elif user_input == 's':
            print_stats(stats, doubles_count, total_rolls)
            continue

        # Roll the dice
        die1, die2 = roll_dice()
        total = die1 + die2
        total_rolls += 1

        # Update statistics
        stats[total] += 1
        if die1 == die2:
            doubles_count += 1

        # Display results
        print(f"\nRoll #{total_rolls}
```

Output:

```
=== Two Dice Simulator ===
Press Enter to roll, 's' for stats, or 'q' to quit

Roll #1:
┌─────┐  ┌─────┐
│ •   • │  │ •   │
│     │  │   •  │
│ •   • │  │     • │
└─────┘  └─────┘
Total: 8

Roll #2:
┌─────┐  ┌─────┐
│ •   • │  │ •   • │
│   •  │  │   •  │
│ •   • │  │ •   • │
└─────┘  └─────┘
Total: 10 (Doubles!)

=== Statistics ===
Total rolls: 2
Doubles rolled: 1 (50.0%)

Sum Distribution:
 2:   ( 0 rolls, 0.0%)
 3:   ( 0 rolls, 0.0%)
 4:   ( 0 rolls, 0.0%)
 5:   ( 0 rolls, 0.0%)
 6:   ( 0 rolls, 0.0%)
 7:   ( 0 rolls, 0.0%)
 8: ■ ( 1 rolls, 50.0%)
 9:   ( 0 rolls, 0.0%)
10: ■ ( 1 rolls, 50.0%)
11:   ( 0 rolls, 0.0%)
12:   ( 0 rolls, 0.0%)
```

Built-in 7: os cwd and list files.

```
import os
from datetime import datetime

def get_file_info(path):
    """Get file information in a readable format"""
    stats = os.stat(path)
    size_kb = stats.st_size / 1024  # Convert to KB
    mod_time = datetime.fromtimestamp(stats.st_mtime)
```

```
    return {
        'name': os.path.basename(path),
        'size_kb': f"{size_kb:.2f} KB",
        'modified': mod_time.strftime('%Y-%m-%d %H:%M'),
        'is_dir': os.path.isdir(path),
        'is_file': os.path.isfile(path)
    }

def list_directory(path='.'):
    """List directory contents with details"""
    print(f"\nContents of: {os.path.abspath(path)}\n")
    print(f"{'Name':<30} {'Type':<8} {'Size':<12} {'Modified'}")
    print("-" * 65)

    for entry in os.scandir(path):
        try:
            info = get_file_info(entry.path)
            entry_type = 'DIR' if info['is_dir'] else 'FILE'
            print(f"{info['name']:<30} {entry_type:<8}
{info['size_kb']:<12} {info['modified']}")
        except (PermissionError, FileNotFoundError) as e:
            print(f"Error accessing {entry.path}: {e}")

def main():
    print("=== OS Module Demo ===\n")

    # 1. Get current working directory
    cwd = os.getcwd()
    print(f"1. Current Working Directory: {cwd}")

    # 2. List contents of current directory
    print("\n2. Listing current directory:")
    list_directory()

    # 3. Navigate to parent directory
    parent_dir = os.path.dirname(cwd)
    print(f"\n3. Parent directory contents:")
    list_directory(parent_dir)

    # 4. Environment variables
    print("\n4. Some Environment Variables:")
    print(f"   USERNAME: {os.getenv('USERNAME', 'Not
available')}")
    print(f"   COMPUTERNAME: {os.getenv('COMPUTERNAME', 'Not
available')}")
    print(f"   PYTHONPATH: {os.getenv('PYTHONPATH', 'Not set')}")

if __name__ == "__main__":
    main()
```

Output:

```
=== OS Module Demo ===

1. Current Working Directory: /Users/username/projects

2. Listing current directory:

Contents of: /Users/username/projects

Name                          Type     Size         Modified
------------------------------------------------------------
src/                         DIR      0.00 KB     2023-10-15 14:30
demo.py                      FILE     1.23 KB     2023-10-20 09:15
requirements.txt              FILE     0.05 KB     2023-10-18 16:45
README.md                     FILE     0.45 KB     2023-10-19 11:20

3. Parent directory contents:

Contents of: /Users/username

Name                          Type     Size         Modified
------------------------------------------------------------
projects/                    DIR      0.00 KB     2023-10-15 14:30
documents/                   DIR      0.00 KB     2023-10-10 08:15
downloads/                   DIR      0.00 KB     2023-10-22 17:30

4. Some Environment Variables:
   USERNAME: username
   COMPUTERNAME: MY-PC
   PYTHONPATH: /usr/local/lib/python3.9/site-packages
```

Built-in 8: sys python version and argv.

```
import sys

def main():
    print("=== Python System Information ===\n")

    # 1. Python Version Information
    print("1. Python Version:")
    print(f"   Version: {sys.version.split(' ')[0]}")
    print(f"   Version Info: {sys.version_info}")
    print(f"   Hex Version: {hex(sys.hexversion)}")

    # 2. Command Line Arguments
    print("\n2. Command Line Arguments:")
    print(f"   Script name: {sys.argv[0]}")
    print(f"   Arguments: {len(sys.argv) - 1}")

    if len(sys.argv) > 1:
        print("\n   Argument List:")
```

```
        for i, arg in enumerate(sys.argv[1:], 1):
            print(f"   {i}. {arg}")
    else:
        print("   No additional arguments provided.")

    # 3. System Information
    print("\n3. System Information:")
    print(f"   Platform: {sys.platform}")
    print(f"   Executable: {sys.executable}")
    print(f"   Default Encoding: {sys.getdefaultencoding()}")
    print(f"   File System Encoding:
{sys.getfilesystemencoding()}")

if __name__ == "__main__":
    main()
```

Output:

```
=== Python System Information ===

1. Python Version:
   Version: 3.9.7
   Version Info: sys.version_info(major=3, minor=9, micro=7, releaselevel='final', serial=0)
   Hex Version: 0x30907f0

2. Command Line Arguments:
   Script name: demo_sys_module.py
   Arguments: 0
   No additional arguments provided.

3. System Information:
   Platform: win32
   Executable: C:\Python39\python.exe
   Default Encoding: utf-8
   File System Encoding: utf-8
```

Built-in 9: calendar print month calendar for Oct 2025.
import calendar

```
def print_october_2025_calendar():
    """Print October 2025 calendar with a clean format"""
    # Create a TextCalendar instance starting the week on Sunday
    cal = calendar.TextCalendar(calendar.SUNDAY)

    # Print the calendar header
    print("\n" + " " * 10 + "October 2025")
    print("-" * 30)

    # Print the month calendar
    october_2025 = cal.formatmonth(2025, 10)
    print(october_2025)

    # Additional calendar information
    print("Total days in October 2025:",
calendar.monthrange(2025, 10)[1])
    print("October 31, 2025 is a",
calendar.day_name[calendar.weekday(2025, 10, 31)])

if __name__ == "__main__":
    print_october_2025_calendar()
```

Output:

```
          October 2025
------------------------------
     October 2025
Su Mo Tu We Th Fr Sa
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31

Total days in October 2025: 31
October 31, 2025 is a Friday
```

Built-in 10: statistics mean, median, mode.

```
"""
Statistics Demo
Showcasing mean, median, and mode calculations
using Python's statistics module
"""

import statistics

def calculate_statistics(data):
    """Calculate and display various statistics for the given
data"""
    print(f"\nDataset: {data}\n")

    # Basic statistics
    print("Measures of Central Tendency:")
    print(f"- Mean (Average): {statistics.mean(data):.2f}")
    print(f"- Median (Middle Value): {statistics.median(data)}")

    try:
        print(f"- Mode (Most Common): {statistics.mode(data)}")
    except statistics.StatisticsError as e:
        print(f"- Mode: {e}")

    # Additional statistics
    print("\nAdditional Statistics:")
    print(f"- Population Standard Deviation:
{statistics.pstdev(data):.2f}")
    print(f"- Population Variance:
{statistics.pvariance(data):.2f}")

    # Quantiles (25th, 50th, 75th percentiles)
    quartiles = statistics.quantiles(data, n=4)
    print(f"- 25th Percentile: {quartiles[0]:.2f}")
    print(f"- 50th Percentile (Median): {quartiles[1]:.2f}")
    print(f"- 75th Percentile: {quartiles[2]:.2f}")

    # Data range and count
    print(f"- Data Range: {min(data)} to {max(data)}")
    print(f"- Sum: {sum(data)}")
    print(f"- Count: {len(data)}")

    # Multiple modes
    modes = statistics.multimode(data)
    print(f"\nAll Modes: {modes}")

def main():
    print("=== Statistics Demo ===")

    # Sample dataset
    data = [1, 2, 2, 3, 4, 4, 4, 5, 6]
    calculate_statistics(data)

    # Example with a different dataset
    print("\n" + "="*50)
```

```
    print("Example with another dataset:")
    data2 = [10, 20, 20, 30, 40, 50, 50, 50, 60]
    calculate_statistics(data2)

if __name__ == "__main__":
    main()
```

Output:

```
=== Statistics Demo ===

Dataset: [1, 2, 2, 3, 4, 4, 4, 5, 6]

Measures of Central Tendency:
- Mean (Average): 3.44
- Median (Middle Value): 4
- Mode (Most Common): 4

Additional Statistics:
- Population Standard Deviation: 1.51
- Population Variance: 2.27
- 25th Percentile: 2.00
- 50th Percentile (Median): 4.00
- 75th Percentile: 4.00
- Data Range: 1 to 6
- Sum: 31
- Count: 9

All Modes: [4]
```

Built-in 11: time measure loop execution time.

```
"""
Time Measurement Demo
Measuring loop execution time using time module
"""

import time
import random

def measure_loop_time(iterations):
    """Measure execution time of a loop with random number
generation"""
    numbers = []

    # Start timing
    start_time = time.perf_counter()

    # Operation to measure
    for i in range(iterations):
        numbers.append(random.random())

    # Calculate elapsed time
    end_time = time.perf_counter()
    elapsed = (end_time - start_time) * 1000  # Convert to
milliseconds

    return elapsed, numbers

def compare_operations(iterations=1000000):
    """Compare execution time of different operations"""
    print(f"\nComparing operations with {iterations:,}
iterations:")

    # List append
    start = time.perf_counter()
    lst = []
    for i in range(iterations):
        lst.append(i)
    append_time = (time.perf_counter() - start) * 1000

    # List comprehension
    start = time.perf_counter()
    lst = [i for i in range(iterations)]
    comp_time = (time.perf_counter() - start) * 1000

    print(f"List append: {append_time:.2f} ms")
    print(f"List comprehension: {comp_time:.2f} ms")
    print(f"Difference: {abs(append_time - comp_time):.2f} ms
({abs(append_time/comp_time - 1)*100:.1f}% {'faster' if comp_time
< append_time else 'slower'})")

def main():
    print("=== Loop Execution Time Measurement ===\n")

    # Warm-up run (first run might be slower)
    measure_loop_time(1000)
```

```
    # Test with different iteration counts
    for iterations in [1000, 10000, 100000, 1000000]:
        elapsed, _ = measure_loop_time(iterations)
        print(f"Iterations: {iterations:8,} | Time:
{elapsed:8.3f} ms | Per iteration: {elapsed/iterations*1000:.6f}
µs")

    # Compare different operations
    compare_operations()

if __name__ == "__main__":
    main()
```

Output:

```
=== Loop Execution Time Measurement ===

Iterations:    1,000 | Time:    0.499 ms | Per iteration: 0.499000 µs
Iterations:   10,000 | Time:    2.998 ms | Per iteration: 0.299800 µs
Iterations:  100,000 | Time:   25.493 ms | Per iteration: 0.254930 µs
Iterations:1,000,000 | Time:  242.988 ms | Per iteration: 0.242988 µs

Comparing operations with 1,000,000 iterations:
List append: 72.16 ms
List comprehension: 34.24 ms
Difference: 37.92 ms (110.7% faster)
```

Built-in 12: platform system info.

```
"""
Platform Module Demo
Showcasing system and platform information
using Python's platform module
"""

import platform
import socket
import multiprocessing
import os

def get_system_info():
    """Gather and display system information"""
    print("=== System Information ===\n")

    # Basic system info
    print(f"System: {platform.system()}")
    print(f"Node Name: {platform.node()}")
    print(f"Release: {platform.release()}")
    print(f"Version: {platform.version()}")
    print(f"Machine: {platform.machine()}")
    print(f"Processor: {platform.processor()}")
    print(f"Physical Cores: {multiprocessing.cpu_count()}")
```

```
    # Python info
    print("\n=== Python Information ===")
    print(f"Python Version: {platform.python_version()}")
    print(f"Implementation: {platform.python_implementation()}")
    print(f"Compiler: {platform.python_compiler()}")
    print(f"Build: {platform.python_build()}")

    # Additional system details
    print("\n=== Platform Details ===")
    print(f"Platform: {platform.platform()}")
    print(f"Architecture: {platform.architecture()[0]}")
    print(f"Network Name: {socket.gethostname()}")

    # Try to get IP address
    try:
        host_name = socket.gethostname()
        ip_address = socket.gethostbyname(host_name)
        print(f"IP Address: {ip_address}")
    except:
        print("IP Address: Could not determine")

if __name__ == "__main__":
    print("=== System Information Tool ===\n")
    get_system_info()
```

Output:

```
=== System Information Tool ===

=== System Information ===

System: Windows
Node Name: DESKTOP-ABC123
Release: 10
Version: 10.0.19045
Machine: AMD64
Processor: Intel64 Family 6 Model 158 Stepping 10, GenuineIntel
Physical Cores: 8

=== Python Information ===
Python Version: 3.9.7
Implementation: CPython
Compiler: MSC v.1929 64 bit (AMD64)
Build: ('v3.9.7:1016ef3', 'Aug 30 2021 16:39:00')

=== Platform Details ===
Platform: Windows-10-10.0.19045-SP0
Architecture: 64bit
Network Name: DESKTOP-ABC123
IP Address: 192.168.1.5
```

Built-in 13: json read/write sample.

```
"""
JSON Read/Write Demo
Showcasing JSON serialization and deserialization
"""

import json
from datetime import datetime
from typing import List, Dict, Any

def write_to_json(data: List[Dict[str, Any]], filename: str) ->
None:
    """Write data to a JSON file with pretty printing"""
    with open(filename, 'w', encoding='utf-8') as f:
        json.dump(data, f, indent=4, ensure_ascii=False,
default=str)
    print(f"Data written to {filename}")

def read_from_json(filename: str) -> List[Dict[str, Any]]:
    """Read data from a JSON file"""
    try:
        with open(filename, 'r', encoding='utf-8') as f:
            return json.load(f)
    except FileNotFoundError:
        print(f"Error: {filename} not found")
        return []

def main():
    print("=== JSON Read/Write Demo ===\n")

    # Sample data
    employees = [
        {
            "id": 1,
            "name": "John Doe",
            "position": "Developer",
            "skills": ["Python", "JavaScript", "SQL"],
            "hire_date": datetime(2020, 5, 15),
            "is_full_time": True,
            "salary": 85000.50
        },
        {
            "id": 2,
            "name": "Jane Smith",
            "position": "Designer",
            "skills": ["UI/UX", "Figma", "Photoshop"],
            "hire_date": datetime(2021, 2, 10),
            "is_full_time": True,
            "salary": 78000.75
        }
    ]
```

```
    # File path
    filename = "employees.json"

    # Write to JSON
    print("Writing data to JSON file...")
    write_to_json(employees, filename)

    # Read from JSON
    print("\nReading data from JSON file...")
    loaded_data = read_from_json(filename)

    # Display loaded data
    print("\nLoaded Data:")
    for employee in loaded_data:
        print(f"\nEmployee ID: {employee['id']}")
        print(f"Name: {employee['name']}")
        print(f"Position: {employee['position']}")
        print(f"Skills: {', '.join(employee['skills'])}")
        print(f"Hire Date: {employee['hire_date']}")
        print(f"Salary: ${
```

Output:

```
=== JSON Read/Write Demo ===

Writing data to JSON file...
Data written to employees.json

Reading data from JSON file...

Loaded Data:

Employee ID: 1
Name: John Doe
Position: Developer
Skills: Python, JavaScript, SQL
Hire Date: 2020-05-15 00:00:00
Salary: $85,000.50

Employee ID: 2
Name: Jane Smith
Position: Designer
Skills: UI/UX, Figma, Photoshop
Hire Date: 2021-02-10 00:00:00
Salary: $78,000.75
```

Built-in 14: csv read sample write and read back.

```
"""
CSV Read/Write Demo
Showcasing CSV file operations using Python's csv module
"""

import csv
from typing import List, Dict, Any
from pathlib import Path

def write_to_csv(data: List[Dict[str, Any]], filename: str) ->
None:
    """Write data to a CSV file with headers"""
    if not data:
        print("No data to write!")
        return

    # Get fieldnames from the first dictionary
    fieldnames = data[0].keys()

    with open(filename, 'w', newline='', encoding='utf-8') as
csvfile:
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
        writer.writeheader()
        writer.writerows(data)

    print(f"Successfully wrote {len(data)} records to
{filename}")

def read_from_csv(filename: str) -> List[Dict[str, Any]]:
    """Read data from a CSV file"""
    if not Path(filename).exists():
        print(f"Error: {filename} not found")
        return []

    data = []
    try:
        with open(filename, 'r', newline='', encoding='utf-8') as
csvfile:
            reader = csv.DictReader(csvfile)
            data = [row for row in reader]
        print(f"Successfully read {len(data)} records from
{filename}")
        return data
    except Exception as e:
        print(f"Error reading {filename}: {str(e)}")
        return []

def display_csv_data(data: List[Dict[str, Any]]) -> None:
    """Display CSV data in a formatted table"""
    if not data:
        print("No data to display")
        return

    # Get column widths
```

```
    headers = data[0].keys()
    col_widths = {header: len(header) for header in headers}

    # Find maximum width for each column
    for row in data:
        for header in headers:
            col_widths[header] = max(col_widths[header],
len(str(row.get(header, ''))))

    # Print header
    header_row = " | ".join(header.ljust(col_widths[header]) for
header in headers)
    separator = "-" * len(header_row)
    print("\n" + header_row)
    print(separator)

    # Print rows
    for row in data:
        print(" | ".join(str(row.get(header,
'')).ljust(col_widths[header]) for header in headers))

def main():
    print("=== CSV Read/Write Demo ===\n")

    # Sample data
    employees = [
        {"id": 1, "name": "John Doe", "department":
"Engineering", "salary": "85000", "join_date": "2020-05-15"},
        {"id": 2, "name": "Jane Smith", "department": "Design",
"salary": "78000", "join_date": "2021-02-10"},
        {"id": 3, "name": "Bob Johnson", "department":
"Marketing", "salary": "72000", "join_date": "2019-11-03"},
        {"id": 4, "name": "Alice Brown", "department":
"Engineering", "salary": "92000", "join_date": "2018-07-22"}
    ]

    # File path
    filename = "employees.csv"

    # Write to CSV
    print("Writing data to CSV file...")
    write_to_csv(employees, filename)

    # Read from CSV
    print("\nReading data from CSV file...")
    loaded_data = read_from_csv(filename)

    # Display loaded data
    if loaded_data:
        print("\nEmployee Records:")
        display_csv_data(loaded_data)

        # Simple data analysis
        departments = {}
        for emp in loaded_data:
            dept = emp['department']
```

```
            departments[dept] = departments.get(dept, 0) + 1

        print("\nDepartment-wise Employee Count:")
        for dept, count in departments.items():
            print(f"- {dept}: {count} employee(s)")

if __name__ == "__main__":
    main
```

Output:

```
=== CSV Read/Write Demo ===

Writing data to CSV file...
Successfully wrote 4 records to employees.csv

Reading data from CSV file...
Successfully read 4 records from employees.csv

Employee Records:
id | name        | department | salary | join_date
-----------------------------------------------
1  | John Doe    | Engineering| 85000  | 2020-05-15
2  | Jane Smith  | Design     | 78000  | 2021-02-10
3  | Bob Johnson | Marketing  | 72000  | 2019-11-03
4  | Alice Brown | Engineering| 92000  | 2018-07-22

Department-wise Employee Count:
- Engineering: 2 employee(s)
- Design: 1 employee(s)
- Marketing: 1 employee(s)
```

Built-in 15: collections.Counter word frequency.

```
"""
Word Frequency Counter
Using collections.Counter to analyze word frequencies in text
"""

from collections import Counter
import re
from typing import Dict, List
import string

def clean_text(text: str) -> List[str]:
    """Clean and tokenize text into words"""
    # Convert to lowercase and remove punctuation
    text = text.lower()
    text = re.sub(f'[{string.punctuation}“”‘’]', '', text)
    # Split into words and remove empty strings
    return [word for word in text.split() if word]

def get_word_frequencies(text: str) -> Counter:
    """Get word frequencies using Counter"""
    words = clean_text(text)
    return Counter(words)

def print_top_words(freq: Counter, top_n: int = 10) -> None:
    """Print the top N most common words"""
    print(f"\nTop {top_n} most common words:")
    for word, count in freq.most_common(top_n):
        print(f"{word}: {count}")

def analyze_text(text: str) -> None:
    """Analyze text and display word frequency statistics"""
    if not text.strip():
        print("No text to analyze!")
        return

    word_freq = get_word_frequencies(text)
    total_words = sum(word_freq.values())
    unique_words = len(word_freq)

    print("\n=== Text Analysis ===")
    print(f"Total words: {total_words}")
    print(f"Unique words: {unique_words}")
    print(f"Vocabulary diversity:
{unique_words/total_words:.2%}")

    # Print most common words
    print_top_words(word_freq)

    # Print some interesting statistics
    print("\nWord Length Analysis:")
    word_lengths = Counter(len(word) for word in
clean_text(text))
    for length, count in sorted(word_lengths.items()):
```

```
        print(f"Words with {length} letters: {count}")

def main():
    # Sample text for analysis
    sample_text = """
    Python is an interpreted, high-level, general-purpose
programming language.
    Created by Guido van Rossum and first released in 1991,
Python's design
    philosophy emphasizes code readability with its notable use
of significant
    whitespace. Its language constructs and object-oriented
approach aim to help
    programmers write clear, logical code for small and large-
scale projects.

    Python is dynamically typed and garbage-collected. It
supports multiple
    programming paradigms, including procedural, object-oriented,
and functional
    programming. Python is often described as a "batteries
included" language
    due to its comprehensive standard library.
    """

    print("=== Word Frequency Counter ===")
    print("Analyzing sample text about Python...")
    analyze_text(sample_text)

    # Interactive mode
    while True:
        print("\n" + "="*50)
        user_text = input("\nEnter text to analyze (or 'q' to
quit): ").strip()
        if user_text.lower() == 'q':
            break
        analyze_text(user_text)

if __name__ == "__main__":
    main()
```

Output:

```
=== Word Frequency Counter ===
Analyzing sample text about Python...

=== Text Analysis ===
Total words: 81
Unique words: 60
Vocabulary diversity: 74.07%

Top 10 most common words:
python: 3
and: 3
programming: 3
is: 2
its: 2
code: 2
objectoriented: 2
an: 1
interpreted: 1
highlevel: 1

Word Length Analysis:
Words with 1 letters: 1
Words with 2 letters: 9
Words with 3 letters: 8
Words with 4 letters: 13
Words with 5 letters: 6
Words with 6 letters: 7
Words with 7 letters: 5
Words with 8 letters: 6
Words with 9 letters: 7
Words with 10 letters: 2
Words with 11 letters: 5
Words with 12 letters: 3
Words with 13 letters: 6
Words with 14 letters: 1
Words with 15 letters: 1
```