

North South University



HW3

https://ece.northsouth.edu/~shahriar.karim/CSE_425/CSE_425_HOMEWORK_4.pdf

Submitted by:

Arka Karmoker

ID: 2112343642

Course: CSE425

Section: 1

Submitted to:

Dr. Md Shahriar Karim [MSK1]

Department of Electrical and Computer Engineering (ECE)

North South University

Date of submission: 25/04/2025

Answer to the question no: 1**problem1_python.py**

```
def result_details():
    # Initialize lists to store course details
    courses = []
    scores = []
    num_courses = 4 # Minimum 4 courses

    # Iterative input collection
    print("Enter details for 4 courses (Course Code, Section, Score):")
    for i in range(num_courses):
        course_code = input(f"Enter Course Code for course {i+1}: ")
        section = input(f"Enter Section for course {i+1}: ")
        while True:
            try:
                score = float(input(f"Enter Score for course {i+1} (0-100): "))
                if 0 <= score <= 100:
                    break
            else:
                print("Score must be between 0 and 100.")
        except ValueError:
            print("Invalid input. Enter a numeric score.")
        courses.append((course_code, section))
        scores.append(score)

    # Calculate average score
    average_score = sum(scores) / len(scores)

    # Display results
    print("\nCourse Details and Scores:")
    for i, (course, score) in enumerate(zip(courses, scores)):
        print(f"Course {i+1}: Code={course[0]}, Section={course[1]}, Score={score}")
    print(f"Average Score: {average_score:.2f}")

# Run the function
if __name__ == "__main__":
    result_details()
```

Output:

Enter details for 4 courses (Course Code, Section, Score):

Enter Course Code for course 1: CSE425

Enter Section for course 1: 1

Enter Score for course 1 (0-100): 80

Enter Course Code for course 2: CSE499A

Enter Section for course 2: 14

Enter Score for course 2 (0-100): 90

Enter Course Code for course 3: CSE440

Enter Section for course 3: 4

Enter Score for course 3 (0-100): 95

Enter Course Code for course 4: EEE452

Enter Section for course 4: 5

Enter Score for course 4 (0-100): 85

Course Details and Scores:

Course 1: Code=CSE425, Section=1, Score=80.0

Course 2: Code=CSE499A, Section=14, Score=90.0

Course 3: Code=CSE440, Section=4, Score=95.0

Course 4: Code=EEE452, Section=5, Score=85.0

Average Score: 87.50

Process finished with exit code 0

The screenshot displays the PyCharm IDE interface. On the left, the 'Project' view shows a file structure with 'problem1.py' selected. The main editor window shows the code for 'problem1.py', which includes a function 'def result_details()' that collects course details and calculates an average score. The code uses loops and conditional statements to handle input validation. On the right, the 'Run' window shows the output of the program, which matches the text provided in the previous blocks. The output includes the prompts for course details, the entered values, the calculated scores, and the final average score of 87.50. The status bar at the bottom indicates the file encoding as UTF-8 and the Python version as 3.10.

```
def result_details():
    # Initialize lists to store course details
    courses = []
    scores = []
    num_courses = 4 # Minimum 4 courses

    # Iterative input collection
    print("Enter details for 4 courses (Course Code, Section, Score):")
    for i in range(num_courses):
        course_code = input(f"Enter Course Code for course {i+1}: ")
        section = input(f"Enter Section for course {i+1}: ")
        while True:
            try:
                score = float(input(f"Enter Score for course {i+1} (0-100): "))
                if 0 <= score <= 100:
                    break
            except ValueError:
                print("Score must be between 0 and 100.")
            except KeyboardInterrupt:
                print("Invalid input. Enter a numeric score.")
        courses.append((course_code, section))
        scores.append(score)

    # Calculate average score
    average_score = sum(scores) / len(scores)

    # Display results
    print(f"Course Details and Scores:")
    for i, (course_code, section) in enumerate(courses, 1):
        score = scores[i-1]
        print(f"Course {i}: Code={course_code}, Section={section}, Score={score}")
    print(f"Average Score: {average_score}")

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

Run: problem1.py x
 "C:\Users\Arka Karemaker\PycharmProjects\HW4\venv\Scripts\python.exe" "C:\Users\Arka Karemaker\PycharmProjects\HW4\problem1.py"
 Enter details for 4 courses (Course Code, Section, Score):
 Enter Course Code for course 1: CSE425
 Enter Section for course 1: 1
 Enter Score for course 1 (0-100): 80
 Enter Course Code for course 2: CSE499A
 Enter Section for course 2: 14
 Enter Score for course 2 (0-100): 90
 Enter Course Code for course 3: CSE440
 Enter Section for course 3: 4
 Enter Score for course 3 (0-100): 95
 Enter Course Code for course 4: EEE452
 Enter Section for course 4: 5
 Enter Score for course 4 (0-100): 85
 Course Details and Scores:
 Course 1: Code=CSE425, Section=1, Score=80.0
 Course 2: Code=CSE499A, Section=14, Score=90.0
 Course 3: Code=CSE440, Section=4, Score=95.0
 Course 4: Code=EEE452, Section=5, Score=85.0
 Average Score: 87.50
 Process finished with exit code 0

problem1_fortran.f90

```

program result_details
  implicit none
  integer, parameter :: num_courses = 4
  character(len=10) :: course_codes(num_courses)
  character(len=5) :: sections(num_courses)
  real :: scores(num_courses)
  real :: average_score
  integer :: i
  character(len=100) :: input_buffer

  print *, "Enter details for 4 courses (Course Code, Section, Score):"
  do i = 1, num_courses
    write(*, '(A,I1,A)') "Enter Course Code for course ", i, ": "
    read(*, '(A)') input_buffer
    course_codes(i) = trim(input_buffer)

    write(*, '(A,I1,A)') "Enter Section for course ", i, ": "
    read(*, '(A)') input_buffer
    sections(i) = trim(input_buffer)

    do
      write(*, '(A,I1,A)') "Enter Score for course ", i, " (0-100): "
      read(*, *, err=10) scores(i)
      if (scores(i) >= 0 .and. scores(i) <= 100) exit
      print *, "Score must be between 0 and 100."
10      print *, "Invalid input. Enter a numeric score."
    end do
  end do

  ! Calculate average score
  average_score = sum(scores) / num_courses

  ! Display results
  print *, ""
  print *, "Course Details and Scores:"
  do i = 1, num_courses
    print '(A,I1,A,A,A,A,A,F5.1)', "Course ", i, ": Code=", trim(course_codes(i)), &
      ", Section=", trim(sections(i)), ", Score=", scores(i)
  end do
  print '(A,F6.2)', "Average Score: ", average_score

end program result_details

```

Output:

Enter details for 4 courses (Course Code, Section, Score):

Enter Course Code for course 1:

CSE425

Enter Section for course 1:

1

Enter Score for course 1 (0-100):

80

Enter Course Code for course 2:

CSE499A

Enter Section for course 2:

14

Enter Score for course 2 (0-100):

90

Enter Course Code for course 3:

CSE440

Enter Section for course 3:

4

Enter Score for course 3 (0-100):

95

Enter Course Code for course 4:

EEE452

Enter Section for course 4:

5

Enter Score for course 4 (0-100):

85

Course Details and Scores:

Course 1: Code=CSE425, Section=1, Score= 80.0

Course 2: Code=CSE499A, Section=14, Score= 90.0

Course 3: Code=CSE440, Section=4, Score= 95.0

Course 4: Code=EEE452, Section=5, Score= 85.0

Average Score: 87.50

Process returned 0 (0x0) execution time : 48.432 s

Press any key to continue.

The screenshot shows a Fortran IDE with a project named 'CSE425' and a file named 'main.f90'. The code in 'main.f90' is as follows:

```

1 program result_details
2   implicit none
3   integer, parameter :: num_courses = 4
4   character(len=10) :: course_codes(num_courses)
5   character(len=5) :: sections(num_courses)
6   real :: scores(num_courses)
7   real :: average_score
8   integer :: i
9   character(len=100) :: input_buffer
10
11   print *, "Enter details for 4 courses (Course Code, Section, Score):"
12   do i = 1, num_courses
13     write(*, '(A,11,A)') "Enter Course Code for course ", i
14     read(*, '(A)') input_buffer
15     course_codes(i) = trim(input_buffer)
16
17     write(*, '(A,11,A)') "Enter Section for course ", i
18     read(*, '(A)') input_buffer
19     sections(i) = trim(input_buffer)
20
21     do
22       write(*, '(A,11,A)') "Enter Score for course ", i
23       read(*, *, err=10) scores(i)
24       if (scores(i) >= 0 .and. scores(i) <= 100) exit
25       print *, "Score must be between 0 and 100."
26     10 continue
27     end do
28   end do

```

The output window shows the execution of the program, where the user enters the following data:

```

Enter details for 4 courses (Course Code, Section, Score):
Enter Course Code for course 1:
CSE425
Enter Section for course 1:
1
Enter Score for course 1 (0-100):
80
Enter Course Code for course 2:
CSE499A
Enter Section for course 2:
14
Enter Score for course 2 (0-100):
90
Enter Course Code for course 3:
CSE440
Enter Section for course 3:
4
Enter Score for course 3 (0-100):
95
Enter Course Code for course 4:
EEE452
Enter Section for course 4:
5
Enter Score for course 4 (0-100):
85

Course Details and Scores:
Course 1: Code=CSE425, Section=1, Score= 80.0
Course 2: Code=CSE499A, Section=14, Score= 90.0
Course 3: Code=CSE440, Section=4, Score= 95.0
Course 4: Code=EEE452, Section=5, Score= 85.0
Average Score: 87.50

```

The process returned 0 (0x0) and the execution time was 58.213 s. The user is prompted to press any key to continue.

Answer to the question no: 2**problem2_python.py**

```
import math

def solve_quadratic():
    # Input coefficients
    try:
        a = float(input("Enter coefficient a: "))
        b = float(input("Enter coefficient b: "))
        c = float(input("Enter coefficient c: "))
    except ValueError:
        print("Invalid input. Coefficients must be numeric.")
        return

    # Check if it's a quadratic equation
    if a == 0:
        print("Error: 'a' cannot be zero for a quadratic equation.")
        return

    # Calculate discriminant
    discriminant = b**2 - 4*a*c

    # Check for complex or real roots
    if discriminant < 0:
        print("Roots are complex.")
        real_part = -b / (2*a)
        imag_part = math.sqrt(-discriminant) / (2*a)
        print(f"Root 1: {real_part:.2f} + {imag_part:.2f}i")
        print(f"Root 2: {real_part:.2f} - {imag_part:.2f}i")
    else:
        root1 = (-b + math.sqrt(discriminant)) / (2*a)
        root2 = (-b - math.sqrt(discriminant)) / (2*a)
        print(f"Root 1: {root1:.2f}")
        print(f"Root 2: {root2:.2f}")

# Run the function
if __name__ == "__main__":
    solve_quadratic()
```

Output 1(Real roots):

Enter coefficient a: 1
 Enter coefficient b: -3
 Enter coefficient c: 2
 Root 1: 2.00
 Root 2: 1.00

Output 2(Complex roots):

Enter coefficient a: 1
 Enter coefficient b: 2
 Enter coefficient c: 5
 Roots are complex.
 Root 1: -1.00 + 2.00i
 Root 2: -1.00 - 2.00i

Output 3(Invalid input):

Enter coefficient a: 0
 Enter coefficient b: 2
 Enter coefficient c: 3
 Error: 'a' cannot be zero for a quadratic equation.

The screenshot shows a PyCharm IDE with a Python script named `problem2.py` and its execution output in the Run console.

Code in `problem2.py`:

```

1  import math
2
3  def solve_quadratic():
4      # Input coefficients
5      try:
6          a = float(input("Enter coefficient a: "))
7          b = float(input("Enter coefficient b: "))
8          c = float(input("Enter coefficient c: "))
9      except ValueError:
10         print("Invalid input. Coefficients must be numeric.")
11         return
12
13     # Check if it's a quadratic equation
14     if a == 0:
15         print("Error: 'a' cannot be zero for a quadratic equation.")
16         return
17
18     # Calculate discriminant
19     discriminant = b**2 - 4*a*c
20
21     # Check for complex or real roots
22     if discriminant < 0:
23         print("Roots are complex.")
24         real_part = -b / (2*a)
25         imag_part = math.sqrt(-discriminant) / (2*a)
26         print(f"Root 1: {real_part:.2f} + {imag_part:.2f}i")

```

Run Console Output:

```

C:\Users\Arka Karimov\PycharmProjects\HW4\venv\Scripts\python.exe "C:\Users\Arka Karimov\PycharmProjects\HW4\problem2.py"
Enter coefficient a: 1
Enter coefficient b: 2
Enter coefficient c: 5
Roots are complex.
Root 1: -1.00 + 2.00i
Root 2: -1.00 - 2.00i

Process finished with exit code 0

```


The screenshot shows the PyCharm IDE with a Python file named `problem2.py`. The code defines a function `solve_quadratic()` that takes coefficients `a`, `b`, and `c` as input. It uses a `try` block to handle `ValueError` exceptions for non-numeric input. The function calculates the discriminant $b^2 - 4ac$ and checks if it is less than zero to determine if the roots are complex or real. For the given input `a=1, b=2, c=5`, the discriminant is $2^2 - 4(1)(5) = 4 - 20 = -16$, which is less than zero, so the roots are complex. The output in the Run console shows: "Enter coefficient a: 1", "Enter coefficient b: 2", "Enter coefficient c: 5", "Roots are complex.", "Root 1: -1.00 + 2.00i", "Root 2: -1.00 - 2.00i", and "Process finished with exit code 0".

```
import math

def solve_quadratic():
    # Input coefficients
    try:
        a = float(input("Enter coefficient a: "))
        b = float(input("Enter coefficient b: "))
        c = float(input("Enter coefficient c: "))
    except ValueError:
        print("Invalid input. Coefficients must be numeric.")
        return

    # Check if it's a quadratic equation
    if a == 0:
        print("Error: 'a' cannot be zero for a quadratic equation.")
        return

    # Calculate discriminant
    discriminant = b**2 - 4*a*c

    # Check for complex or real roots
    if discriminant < 0:
        print("Roots are complex.")
        real_part = -b / (2*a)
        imag_part = math.sqrt(-discriminant) / (2*a)
        print(f"Root 1: {real_part:.2f} + {imag_part:.2f}i")
        print(f"Root 2: {real_part:.2f} - {imag_part:.2f}i")
    else:
        # Real roots
        root1 = (-b + math.sqrt(discriminant)) / (2*a)
        root2 = (-b - math.sqrt(discriminant)) / (2*a)
        print(f"Root 1: {root1:.2f}")
        print(f"Root 2: {root2:.2f}")

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

The screenshot shows the same PyCharm IDE with `problem2.py`, but the input values are `a=0, b=2, c=3`. Since `a=0`, the function prints an error message: "Error: 'a' cannot be zero for a quadratic equation." The Run console output shows: "Enter coefficient a: 0", "Enter coefficient b: 2", "Enter coefficient c: 3", "Error: 'a' cannot be zero for a quadratic equation.", and "Process finished with exit code 0".

```
import math

def solve_quadratic():
    # Input coefficients
    try:
        a = float(input("Enter coefficient a: "))
        b = float(input("Enter coefficient b: "))
        c = float(input("Enter coefficient c: "))
    except ValueError:
        print("Invalid input. Coefficients must be numeric.")
        return

    # Check if it's a quadratic equation
    if a == 0:
        print("Error: 'a' cannot be zero for a quadratic equation.")
        return

    # Calculate discriminant
    discriminant = b**2 - 4*a*c

    # Check for complex or real roots
    if discriminant < 0:
        print("Roots are complex.")
        real_part = -b / (2*a)
        imag_part = math.sqrt(-discriminant) / (2*a)
        print(f"Root 1: {real_part:.2f} + {imag_part:.2f}i")
        print(f"Root 2: {real_part:.2f} - {imag_part:.2f}i")
    else:
        # Real roots
        root1 = (-b + math.sqrt(discriminant)) / (2*a)
        root2 = (-b - math.sqrt(discriminant)) / (2*a)
        print(f"Root 1: {root1:.2f}")
        print(f"Root 2: {root2:.2f}")

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

problem2_fortran.f90

```

program solve_quadratic
  implicit none
  real :: a, b, c, discriminant, root1, root2, real_part, imag_part

  ! Input coefficients
  print *, "Enter coefficient a: "
  read(*, *, err=10) a
  print *, "Enter coefficient b: "
  read(*, *, err=10) b
  print *, "Enter coefficient c: "
  read(*, *, err=10) c

  ! Check if it's a quadratic equation
  if (a == 0) then
    print *, "Error: 'a' cannot be zero for a quadratic equation."
    stop
  end if

  ! Calculate discriminant
  discriminant = b**2 - 4*a*c

  ! Check for complex or real roots
  if (discriminant < 0) then
    print *, "Roots are complex."
    real_part = -b / (2*a)
    imag_part = sqrt(-discriminant) / (2*a)
    print '(A,F6.2,A,F6.2,A)', "Root 1: ", real_part, " + ", imag_part, "i"
    print '(A,F6.2,A,F6.2,A)', "Root 2: ", real_part, " - ", imag_part, "i"
  else
    root1 = (-b + sqrt(discriminant)) / (2*a)
    root2 = (-b - sqrt(discriminant)) / (2*a)
    print '(A,F6.2)', "Root 1: ", root1
    print '(A,F6.2)', "Root 2: ", root2
  end if
  stop

10 print *, "Invalid input. Coefficients must be numeric."
end program solve_quadratic

```

Output 1(Real roots):

Enter coefficient a:

1

Enter coefficient b:

-3

Enter coefficient c:

2

Root 1: 2.00

Root 2: 1.00

Process returned 0 (0x0) execution time : 9.100 s

Press any key to continue.

Output 2(Complex roots):

Enter coefficient a:

1

Enter coefficient b:

2

Enter coefficient c:

5

Roots are complex.

Root 1: -1.00 + 2.00i

Root 2: -1.00 - 2.00i

Process returned 0 (0x0) execution time : 6.543 s

Press any key to continue.

Output 3(Invalid input):

Enter coefficient a:

0

Enter coefficient b:

2

Enter coefficient c:

3

Error: 'a' cannot be zero for a quadratic equation.

Process returned 0 (0x0) execution time : 3.009 s

Press any key to continue.

The screenshot shows the Code::Blocks IDE with a Fortran project named 'CSE425'. The main window displays the source code for 'main.f90', which is a program to solve a quadratic equation. The code prompts the user to enter coefficients a, b, and c. It checks if a is zero, calculates the discriminant, and then determines if the roots are complex or real. In this case, the roots are real. The output window shows the execution results: 'Enter coefficient a: 1', 'Enter coefficient b: -3', 'Enter coefficient c: 2', 'Root 1: 2.00', 'Root 2: 1.00', 'Process returned 0 (0x0) execution time : 9.100 s', and 'Press any key to continue.'.

```

1 program solve quadratic
2   implicit none
3   real :: a, b, c, discriminant, root1, root2, real_part, im
4
5   ! Input coefficients
6   print *, "Enter coefficient a: "
7   read(*, *, err=10) a
8   print *, "Enter coefficient b: "
9   read(*, *, err=10) b
10  print *, "Enter coefficient c: "
11  read(*, *, err=10) c
12
13  ! Check if it's a quadratic equation
14  if (a == 0) then
15    print *, "Error: 'a' cannot be zero for a quadratic eq"
16    stop
17  end if
18
19  ! Calculate discriminant
20  discriminant = b**2 - 4*a*c
21
22  ! Check for complex or real roots
23  if (discriminant < 0) then
24    print *, "Roots are complex."
25    real_part = -b / (2*a)
26    imag_part = sqrt(-discriminant) / (2*a)
27    print '(A,E6.2,A,E6.2,A)', "Root 1: ", real_part, " + "
28    print '(A,E6.2,A,E6.2,A)', "Root 2: ", real_part, " - "

```

The screenshot shows the Code::Blocks IDE with the same Fortran project 'CSE425'. The main window displays the source code for 'main.f90', which is identical to the previous one. The output window shows the execution results for a different set of coefficients: 'Enter coefficient a: 1', 'Enter coefficient b: 1', 'Enter coefficient c: 5', 'Roots are complex.', 'Root 1: -1.00 + 2.00i', 'Root 2: -1.00 - 2.00i', 'Process returned 0 (0x0) execution time : 6.543 s', and 'Press any key to continue.'.

```

1 program solve quadratic
2   implicit none
3   real :: a, b, c, discriminant, root1, root2, real_part, im
4
5   ! Input coefficients
6   print *, "Enter coefficient a: "
7   read(*, *, err=10) a
8   print *, "Enter coefficient b: "
9   read(*, *, err=10) b
10  print *, "Enter coefficient c: "
11  read(*, *, err=10) c
12
13  ! Check if it's a quadratic equation
14  if (a == 0) then
15    print *, "Error: 'a' cannot be zero for a quadratic eq"
16    stop
17  end if
18
19  ! Calculate discriminant
20  discriminant = b**2 - 4*a*c
21
22  ! Check for complex or real roots
23  if (discriminant < 0) then
24    print *, "Roots are complex."
25    real_part = -b / (2*a)
26    imag_part = sqrt(-discriminant) / (2*a)
27    print '(A,E6.2,A,E6.2,A)', "Root 1: ", real_part, " + "
28    print '(A,E6.2,A,E6.2,A)', "Root 2: ", real_part, " - "

```

The screenshot displays the Code::Blocks IDE with a Fortran project named 'CSE425'. The main window shows the source code for a program called 'solve quadratic'. The code prompts the user to enter coefficients a, b, and c. It includes a check for a=0, which triggers an error message. The program also calculates the discriminant and prints the roots if they are real. The output window on the right shows the program's execution, including the error message and the execution time of 3.009 seconds.

```

1 program solve quadratic
2   implicit none
3   real :: a, b, c, discriminant, root1, root2, real_part, im
4
5   ! Input coefficients
6   print *, "Enter coefficient a: "
7   read(*, *, err=10) a
8   print *, "Enter coefficient b: "
9   read(*, *, err=10) b
10  print *, "Enter coefficient c: "
11  read(*, *, err=10) c
12
13  ! Check if it's a quadratic equation
14  if (a == 0) then
15    print *, "Error: 'a' cannot be zero for a quadratic eq"
16    stop
17  end if
18
19  ! Calculate discriminant
20  discriminant = b**2 - 4*a*c
21
22  ! Check for complex or real roots
23  if (discriminant < 0) then
24    print *, "Roots are complex."
25    real_part = -b / (2*a)
26    imag_part = sqrt(-discriminant) / (2*a)
27    print '(A,E6.2,A,E6.2,A)', "Root 1: ", real_part, " + "
28    print '(A,E6.2,A,E6.2,A)', "Root 2: ", real_part, " - "

```

Output window content:

```

Enter coefficient a:
Enter coefficient b:
Enter coefficient c:
Error: 'a' cannot be zero for a quadratic equation.
Process returned 0 (0x0)   execution time : 3.009 s
Press any key to continue.

```

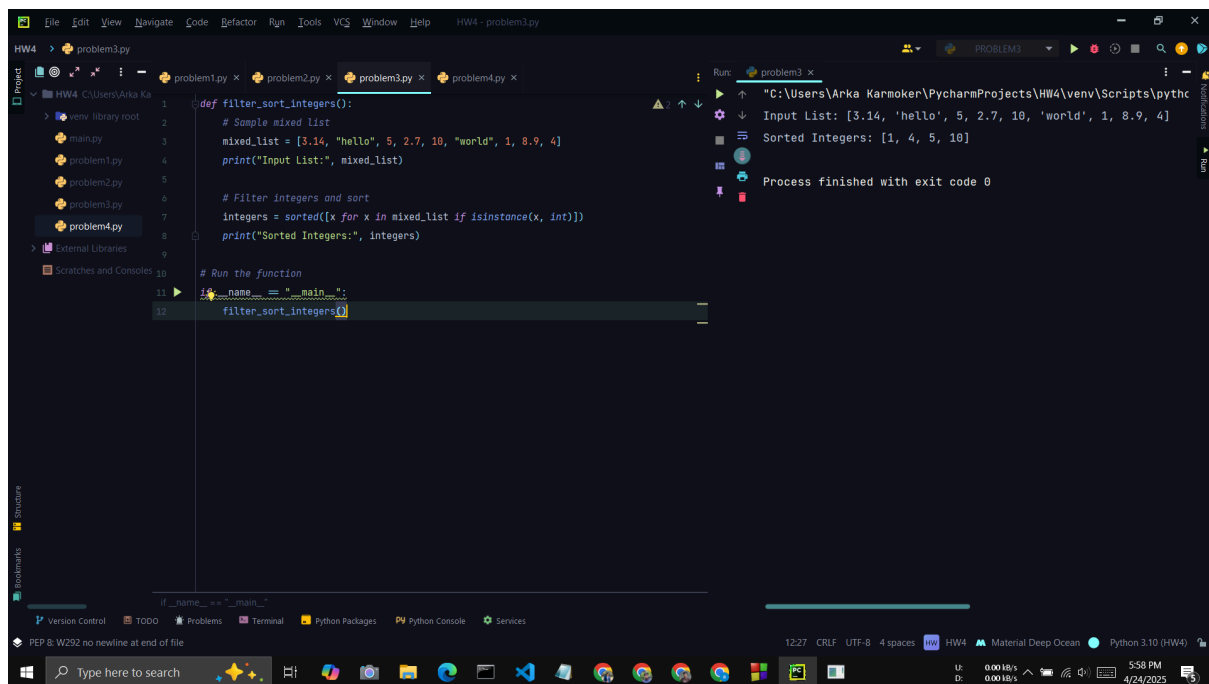
Answer to the question no: 3

problem3_python.py

```
def filter_sort_integers():  
    # Sample mixed list  
    mixed_list = [3.14, "hello", 5, 2.7, 10, "world", 1, 8.9, 4]  
    print("Input List:", mixed_list)  
  
    # Filter integers and sort  
    integers = sorted([x for x in mixed_list if isinstance(x, int)])  
    print("Sorted Integers:", integers)  
  
# Run the function  
if __name__ == "__main__":  
    filter_sort_integers()
```

Output:

Input List: [3.14, 'hello', 5, 2.7, 10, 'world', 1, 8.9, 4]
Sorted Integers: [1, 4, 5, 10]



problem3_fortran.f90

```

program filter_sort_integers
  implicit none
  integer, parameter :: n = 9
  integer :: i, j, temp, int_count
  real :: mixed_list(n) = [3.14, 5.0, 2.7, 10.0, 1.0, 8.9, 4.0, 6.0, 7.5]
  integer, allocatable :: integers(:)

  ! Print input list
  print *, "Input List:"
  print '(9F6.2)', mixed_list

  ! Count integers (numbers with no decimal part)
  int_count = 0
  do i = 1, n
    if (mixed_list(i) == floor(mixed_list(i))) then
      int_count = int_count + 1
    end if
  end do

  ! Allocate array for integers
  allocate(integers(int_count))

  ! Extract integers
  j = 1
  do i = 1, n
    if (mixed_list(i) == floor(mixed_list(i))) then
      integers(j) = int(mixed_list(i))
      j = j + 1
    end if
  end do

  ! Sort integers (Bubble Sort)
  do i = 1, int_count-1
    do j = 1, int_count-i
      if (integers(j) > integers(j+1)) then
        temp = integers(j)
        integers(j) = integers(j+1)
        integers(j+1) = temp
      end if
    end do
  end do

  ! Print sorted integers
  print *, "Sorted Integers:"
  print '(10I5)', integers

```


Answer to the question no: 4**problem4_python.py**

```
import random
import math

def cumulative_sum_and_circle_area():
    # Generate list of 10 random numbers (integers for simplicity)
    random_list = [random.randint(1, 20) for _ in range(10)]
    print("Random List:", random_list)

    # Compute cumulative sum
    cumulative_list = [sum(random_list[:i + 1]) for i in range(len(random_list))]
    print("Cumulative Sum List:", cumulative_list)

    # Find numbers divisible by 3 and calculate circle area
    areas = [math.pi * (x ** 2) for x in cumulative_list if x % 3 == 0]

    if areas:
        print("Areas of circles (radius divisible by 3):")
        for i, area in enumerate(areas, 1):
            print(f"Area {i}: {area:.2f}")
    else:
        print("radius not found")

# Run the function
if __name__ == "__main__":
    # random.seed(42) # For reproducibility
    cumulative_sum_and_circle_area()
```

Output:

Random List: [4, 3, 15, 18, 11, 13, 11, 8, 2, 5]

Cumulative Sum List: [4, 7, 22, 40, 51, 64, 75, 83, 85, 90]

Areas of circles (radius divisible by 3):

Area 1: 8171.28

Area 2: 17671.46

Area 3: 25446.90

The screenshot shows a PyCharm IDE with a Python script named `problem4.py` and its execution output. The script generates a random list of 10 integers, calculates their cumulative sum, and then finds the areas of circles for radii divisible by 3. The output window displays the results of these operations.

```

1  import random
2  import math
3
4  def cumulative_sum_and_circle_area():
5      # Generate list of 10 random numbers (integers for simplicity)
6      random_list = [random.randint(1, 20) for _ in range(10)]
7      print("Random List:", random_list)
8
9      # Compute cumulative sum
10     cumulative_list = [sum(random_list[:i + 1]) for i in range(len(random_list))]
11     print("Cumulative Sum List:", cumulative_list)
12
13     # Find numbers divisible by 3 and calculate circle area
14     areas = [math.pi * (x ** 2) for x in cumulative_list if x % 3 == 0]
15
16     if areas:
17         print("Areas of circles (radius divisible by 3):")
18         for i, area in enumerate(areas, 1):
19             print(f"Area {i}: {area:.2f}")
20     else:
21         print("radius not found")
22
23 # Run the function
24 if __name__ == "__main__":
25     # random.seed(42) # For reproducibility
26     cumulative_sum_and_circle_area()

```

Output:

```

"C:\Users\Arka Karmoker\PycharmProjects\HW4\venv\Scripts\python.exe" "C:\Users\Arka Karmoker\PycharmProjects\HW4\problem4.py"
Random List: [4, 3, 15, 18, 11, 13, 11, 8, 2, 5]
Cumulative Sum List: [4, 7, 22, 40, 51, 64, 75, 83, 85, 90]
Areas of circles (radius divisible by 3):
Area 1: 8171.28
Area 2: 17671.46
Area 3: 25446.90

Process finished with exit code 0

```

problem4_fortran.f90

```

program cumulative_sum_and_circle_area
  implicit none
  integer, parameter :: n = 10
  real :: random_list(n), cumulative_list(n)
  real :: pi = 3.14159265359
  integer :: i, count
  logical :: found

  ! Seed for reproducibility
  call random_seed()

  ! Generate random numbers (1 to 20)
  do i = 1, n
    call random_number(random_list(i))
    random_list(i) = 1 + floor(20 * random_list(i))
  end do

  ! Print random list
  print *, "Random List:"
  print '(10I5)', int(random_list)

  ! Compute cumulative sum
  cumulative_list(1) = random_list(1)
  do i = 2, n
    cumulative_list(i) = cumulative_list(i-1) + random_list(i)
  end do

  ! Print cumulative sum list
  print *, "Cumulative Sum List:"
  print '(10I5)', int(cumulative_list)

  ! Calculate circle areas for numbers divisible by 3
  found = .false.
  print *, "Areas of circles (radius divisible by 3):"
  count = 0
  do i = 1, n
    if (mod(int(cumulative_list(i)), 3) == 0) then
      found = .true.
      count = count + 1
      print '(A,I2,A,F10.2)', "Area ", count, ": ", pi * (cumulative_list(i) ** 2)
    end if
  end do

  if (.not. found) then
    print *, "radius not found"
  end if

```

end program cumulative_sum_and_circle_area

Output:

Random List:

4 16 11 4 16 9 12 1 6 15

Cumulative Sum List:

4 20 31 35 51 60 72 73 79 94

Areas of circles (radius divisible by 3):

Area 1: 8171.28

Area 2: 11309.73

Area 3: 16286.02

Process returned 0 (0x0) execution time : 0.021 s

Press any key to continue.

```

program cumulative_sum_and_circle_area
  implicit none
  integer, parameter :: n = 10
  real :: random_list(n), cumulative_list(n)
  real :: pi = 3.14159265359
  integer :: i, count
  logical :: found

  ! Seed for reproducibility
  call random_seed()

  ! Generate random numbers (1 to 20)
  do i = 1, n
    call random_number(random_list(i))
    random_list(i) = 1 + floor(20 * random_list(i))
  end do

  ! Print random list
  print *, "Random List:"
  print '(10i5)', int(random_list)

  ! Compute cumulative sum
  cumulative_list(1) = random_list(1)
  do i = 2, n
    cumulative_list(i) = cumulative_list(i-1) + random_list(i)
  end do

  ! Print cumulative sum list
  print *, "Cumulative Sum List:"
  print '(10i5)', int(cumulative_list)

  ! Areas of circles (radius divisible by 3)
  count = 0
  do i = 1, n
    if (mod(random_list(i), 3) == 0) then
      count = count + 1
    end if
  end do

  ! Print areas of circles
  print *, "Areas of circles (radius divisible by 3):"
  do i = 1, count
    print *, "Area ", i, ": ", pi * (random_list(i)**2)
  end do

  print *, "Process returned 0 (0x0) execution time : 0.021 s"
  print *, "Press any key to continue."
end program cumulative_sum_and_circle_area
  
```

Random List:
4 16 11 4 16 9 12 1 6 15
Cumulative Sum List:
4 20 31 35 51 60 72 73 79 94
Areas of circles (radius divisible by 3):
Area 1: 8171.28
Area 2: 11309.73
Area 3: 16286.02
Process returned 0 (0x0) execution time : 0.021 s
Press any key to continue.