

AI ASSISTED CODING LAB ASSIGNMENT -13.3

NAME : S.NADHIYA

BATCH : 05

ROLLNO : 2403A510C6

DEPT : CSE

Task – 01

Provide AI with the following redundant code and ask it to refactor

Prompt:

Refactor the following Python code to make it cleaner and more modular.

Use **dictionary-based dispatch** or **separate functions** for each shape instead of multiple if-elif statements.

Also, allow the user to provide input for shape and dimensions at runtime.

Make sure the code is easy to extend with more shapes in the future.

```
def calculate_area(shape, x, y=0):
```

```
    if shape == "rectangle":
```

```
        return x * y
```

```
    elif shape == "square":
```

```
        return x * x
```

```
    elif shape == "circle":
```

```
        return 3.14 * x * x
```

Code generated:

```
area_calculator.py > ...
1  """
2  Area Calculator - A modular and extensible area calculation system.
3  Supports multiple shapes with dictionary-based dispatch pattern.
4  """
5
6  import math
7  from typing import Dict, Callable, Any
8
9
10 class AreaCalculator:
11     """A calculator for computing areas of various shapes."""
12
13     def __init__(self):
14         """Initialize the calculator with shape handlers."""
15         self.shape_handlers: Dict[str, Callable] = {
16             'rectangle': self._calculate_rectangle,
17             'square': self._calculate_square,
18             'circle': self._calculate_circle,
19             'triangle': self._calculate_triangle,
20             'ellipse': self._calculate_ellipse
21         }
22
23     def _calculate_rectangle(self, length: float, width: float) -> float:
24         """Calculate area of rectangle."""
25         return length * width
26
27     def _calculate_square(self, side: float) -> float:
28         """Calculate area of square."""
29         return side * side
30
31     def _calculate_circle(self, radius: float) -> float:
32         """Calculate area of circle."""
33         return math.pi * radius * radius
34
35     def _calculate_triangle(self, base: float, height: float) -> float:
36         """Calculate area of triangle."""
37         return 0.5 * base * height
38
39     def _calculate_ellipse(self, a: float, b: float) -> float:
40         """Calculate area of ellipse."""
41         return math.pi * a * b
42
43     def calculate_area(self, shape: str, *dimensions: float) -> float:
```

```
area_calculator.py > ...
10 class AreaCalculator:
43     def calculate_area(self, shape: str, *dimensions: float) -> float:
44         """
45         Calculate area for given shape and dimensions.
46
47         Args:
48             shape: Name of the shape
49             *dimensions: Variable number of dimension values
50
51         Returns:
52             Area of the shape
53
54         Raises:
55             ValueError: If shape is not supported or dimensions are invalid
56         """
57         if shape not in self.shape_handlers:
58             raise ValueError(f"Unsupported shape: {shape}")
59
60         if not dimensions or any(d <= 0 for d in dimensions):
61             raise ValueError("All dimensions must be positive numbers")
62
63         return self.shape_handlers[shape](*dimensions)
64
65     def get_supported_shapes(self) -> list:
66         """Return list of supported shapes."""
67         return list(self.shape_handlers.keys())
68
69     def add_shape(self, name: str, handler: Callable) -> None:
70         """Add a new shape handler for extensibility."""
71         self.shape_handlers[name] = handler
72
73
74     def get_user_input() -> tuple:
75         """Get shape and dimensions from user input."""
76         print("Available shapes:", ', '.join(AreaCalculator().get_supported_shapes()))
77
78         shape = input("Enter shape: ").lower().strip()
79
80         if shape == 'rectangle':
81             length = float(input("Enter length: "))
82             width = float(input("Enter width: "))
83             return shape, length, width
84         elif shape == 'square':
85             side = float(input("Enter side length: "))
86             return shape, side, side
```

```

74 def get_user_input() -> tuple:
75     side = float(input("Enter side length: "))
76     return shape, side
77 elif shape == 'circle':
78     radius = float(input("Enter radius: "))
79     return shape, radius
80 elif shape == 'triangle':
81     base = float(input("Enter base: "))
82     height = float(input("Enter height: "))
83     return shape, base, height
84 elif shape == 'ellipse':
85     a = float(input("Enter semi-major axis: "))
86     b = float(input("Enter semi-minor axis: "))
87     return shape, a, b
88 else:
89     raise ValueError(f"Unsupported shape: {shape}")
90
91
92
93
94
95
96
97
98
99
100
101
102 def main():
103     """Main function to run the area calculator."""
104     calculator = AreaCalculator()
105
106     try:
107         shape, *dimensions = get_user_input()
108         area = calculator.calculate_area(shape, *dimensions)
109         print(f"Area of {shape}: {area:.2f}")
110     except ValueError as e:
111         print(f"Error: {e}")
112     except KeyboardInterrupt:
113         print("\nProgram interrupted by user.")
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

Output:

```

Problems Output Debug Console Terminal Ports
PS C:\Users\nadhi\OneDrive\Desktop\AI> & C:/Users/nadhi/anaconda3/python.exe c:/Users/nadhi/OneDrive/Desktop/AI/area_calculator.py
Available shapes: rectangle, square, circle, triangle, ellipse
Enter shape: circle
Enter radius: 5
Area of circle: 78.54
PS C:\Users\nadhi\OneDrive\Desktop\AI>

```

Observation:

Dictionary-based dispatch: Cleaner than multiple if-elif statements.

Separate functions: Each shape has its own function, making it modular and easy to extend. **User input handling:** Program asks the user what shape and dimensions to calculate. **Error handling:** Gracefully handles invalid shapes.

Task -02

Legacy function without proper error handling Python Code

Prompt:

Refactor the following legacy Python function to improve safety and readability:

- Use with open() context manager to handle files.
- Add proper try-except error handling for cases where the file does not exist or cannot be read.
- Return a helpful message if an error occurs.

```
def read_file(filename):
```

```
    f = open(filename, "r")
```

```
    data = f.read()
```

```
    f.close()
```

```
    return data
```

Code generated:

```

> refactored_read_file.py > read_file
1 def read_file(filename):
2     """
3     Read the contents of a file and return the data.
4
5     Args:
6     | filename (str): The path to the file to read
7
8     Returns:
9     | str: The contents of the file, or an error message if the file cannot be read
10
11     Example:
12     | >>> content = read_file("example.txt")
13     | >>> print(content)
14     """
15     try:
16         with open(filename, "r", encoding="utf-8") as file:
17             data = file.read()
18             return data
19     except FileNotFoundError:
20         return f"Error: The file '{filename}' was not found."
21     except PermissionError:
22         return f"Error: Permission denied. Cannot read the file '{filename}'."
23     except UnicodeDecodeError:
24         return f"Error: Cannot decode the file '{filename}'. The file may contain invalid characters."
25     except IOError as e:
26         return f"Error: Unable to read the file '{filename}'. {str(e)}"
27     except Exception as e:
28         return f"Error: An unexpected error occurred while reading '{filename}': {str(e)}"
29
30
31 # Example usage and testing
32 if __name__ == "__main__":
33     # Test with an existing file
34     print("Testing with existing file:")
35     result = read_file("refactored_read_file.py")
36     print(f"File read successfully: {len(result)} characters")
37
38     # Test with non-existent file
39     print("\nTesting with non-existent file:")

```

Problems Output Debug Console Terminal Ports

```

> refactored_read_file.py > read_file
1 def read_file(filename):
2     """
3     Read the contents of a file and return the data.
4
5     Args:
6     | filename (str): The path to the file to read
7
8     Returns:
9     | str: The contents of the file, or an error message if the file cannot be read
10
11     Example:
12     | >>> content = read_file("example.txt")
13     | >>> print(content)
14     """
15     try:
16         with open(filename, "r", encoding="utf-8") as file:
17             data = file.read()
18             return data
19     except FileNotFoundError:
20         return f"Error: The file '{filename}' was not found."
21     except PermissionError:
22         return f"Error: Permission denied. Cannot read the file '{filename}'."
23     except UnicodeDecodeError:
24         return f"Error: Cannot decode the file '{filename}'. The file may contain invalid characters."
25     except IOError as e:
26         return f"Error: Unable to read the file '{filename}'. {str(e)}"
27     except Exception as e:
28         return f"Error: An unexpected error occurred while reading '{filename}': {str(e)}"
29
30
31 # Example usage and testing
32 if __name__ == "__main__":
33     # Test with an existing file
34     print("Testing with existing file:")
35     result = read_file("refactored_read_file.py")
36     print(f"File read successfully: {len(result)} characters")
37
38     # Test with non-existent file
39     print("\nTesting with non-existent file:")

```

Problems Output Debug Console Terminal Ports

```

refactored_read_file.py > read_file
1  def read_file(filename):
2      except PermissionError:
21         return f"Error: Permission denied. Cannot read the file '{filename}'."
22
23     except UnicodeDecodeError:
24         return f"Error: Cannot decode the file '{filename}'. The file may contain invalid characters."
25
26     except IOError as e:
27         return f"Error: Unable to read the file '{filename}'. {str(e)}"
28
29     except Exception as e:
30         return f"Error: An unexpected error occurred while reading '{filename}': {str(e)}"
31
32 # Example usage and testing
33 if __name__ == "__main__":
34     # Test with an existing file
35     print("Testing with existing file:")
36     result = read_file("refactored_read_file.py")
37     print(f"File read successfully: {len(result)} characters")
38
39     # Test with non-existent file
40     print("\nTesting with non-existent file:")
41     result = read_file("nonexistent.txt")
42     print(result)
43
44     # Test with a file that might have permission issues (if applicable)
45     print("\nTesting with protected file:")
46     result = read_file("C:\\Windows\\System32\\config\\SAM") # Usually protected
47     print(result)

```

Output:

```

PS C:\Users\nadhi\OneDrive\Desktop\AI> & C:/Users/nadhi/anaconda3/python.exe c:/Users/nadhi/OneDrive/Desktop/AI/refactored_read_file.py
Testing with existing file:
File read successfully: 1646 characters

Testing with non-existent file:
Error: The file 'nonexistent.txt' was not found.

Testing with protected file:
Error: Permission denied. Cannot read the file 'C:\Windows\System32\config\SAM'.
PS C:\Users\nadhi\OneDrive\Desktop\AI> 

```

Observation:

Docstring is clear and includes arguments, return type, and example usage.

with open() ensures the file is automatically closed (even if an exception occurs).

Granular exception handling (FileNotFoundError, PermissionError, UnicodeDecodeError) improves usability.

Fallback Exception handler catches any unexpected issues.

Test cases are included under `if __name__ == "__main__":` – good practice.

Task -03

Provide this legacy class to AI for readability and modularity improvements

Prompt:

Refactor the following legacy Python class to improve **readability** and **modularity**:

- Use **descriptive attribute names** (name, age, marks instead of n, a, m1, m2, m3).
- Store marks in a **list** so that `sum(self.marks)` can be used to calculate total.
- Add a **docstring** to the class and methods.
- Improve the `details()` method output for better readability (e.g., formatted string).
- Allow the user to **input details** (name, age, marks) at runtime and display the details + total marks.

class Student:

```
def __init__(self, n, a, m1, m2, m3):
```

```
    self.n = n
```

```
    self.a = a
```

```
    self.m1 = m1
```

```
    self.m2 = m2
```

```
    self.m3 = m3
```

```
def details(self):
```

```
    print("Name:", self.n, "Age:", self.a)
```

```
def total(self):
```

```
    return self.m1 + self.m2 + self.m3
```

Code generated:

refactored_student.py > ...

```
1  """
2  Student Management System
3
4  This module contains a refactored Student class with improved readability,
5  modularity, and functionality for managing student information.
6  """
7
8
9  class Student:
10     """
11     A class to represent a student with their personal information and academic marks.
12
13     Attributes:
14         name (str): The student's full name
15         age (int): The student's age
16         marks (list): A list of the student's marks for different subjects
17     """
18
19     def __init__(self, name, age, marks):
20         """
21         Initialize a new Student instance.
22
23         Args:
24             name (str): The student's full name
25             age (int): The student's age
26             marks (list): A list of marks for different subjects
27         """
28         self.name = name
29         self.age = age
30         self.marks = marks
31
32     def details(self):
33         """
34         Display the student's basic information in a formatted way.
35
36         Returns:
37             str: Formatted string containing student details
38         """
39         details_str = f"Student Details:\n"
40         details_str += f"Name: {self.name}\n"
41         details_str += f"Age: {self.age}\n"
42         details_str += f"Marks: {' '.join(map(str, self.marks))}\n"
43         details_str += f"Total Marks: {self.total()}"
44         return details_str
```

```

refactored_student.py > ...
9  class Student:
10      """
11      """
12      return details_str
13
14
15
16  def total(self):
17      """
18      Calculate the total marks for the student.
19
20      Returns:
21          int: Sum of all marks
22      """
23      return sum(self.marks)
24
25
26  def average(self):
27      """
28      Calculate the average marks for the student.
29
30      Returns:
31          float: Average of all marks
32      """
33      return sum(self.marks) / len(self.marks) if self.marks else 0
34
35
36
37  def get_student_input():
38      """
39      Get student information from user input.
40
41      Returns:
42          Student: A new Student instance with user-provided data
43      """
44      print("Enter Student Information:")
45      print("-" * 30)
46
47      # Get student name
48      name = input("Enter student name: ").strip()
49      while not name:
50          print("Name cannot be empty. Please try again.")
51          name = input("Enter student name: ").strip()
52
53      # Get student age
54      while True:
55          try:
56              age = int(input("Enter student age: "))
57              if age < 0:
58                  print("Age cannot be negative. Please try again.")

```

```
refactored_student.py
refactored_student.py > ...
65 def get_student_input():
93     marks = []
94     print("Enter marks for each subject (press Enter with empty input to finish):")
95     subject_num = 1
96
97     while True:
98         try:
99             mark_input = input(f"Enter mark for subject {subject_num} (or press Enter to finish): ").strip()
100             if not mark_input:
101                 break
102             mark = float(mark_input)
103             if mark < 0 or mark > 100:
104                 print("Marks should be between 0 and 100. Please try again.")
105                 continue
106             marks.append(mark)
107             subject_num += 1
108         except ValueError:
109             print("Please enter a valid mark (number).")
110
111     if not marks:
112         print("No marks entered. Using default marks [0, 0, 0]")
113         marks = [0, 0, 0]
114
115     return Student(name, age, marks)
116
117
118 def main():
119     """
120     Main function to demonstrate the refactored Student class.
121     """
122     print("Student Management System")
123     print("=" * 50)
124
125     # Create student from user input
126     student = get_student_input()
127
128     print("\n" + "=" * 50)
129     print(student.details())
130     print(f"Average Marks: {student.average():.2f}")
131     print("=" * 50)
132
133
134 if __name__ == "__main__":
135     main()

Problems Output Debug Console Terminal Ports
```

Output:

```
PS C:\Users\nadhi\OneDrive\Desktop\AI> & C:/Users/nadhi/anaconda3/python.exe c:/Users/nadhi/OneDrive/Desktop/AI/refactored_student.py
Student Management System
=====
Enter Student Information:
-----
Student Management System
=====
Enter Student Information:
-----
Enter Student Information:
-----
Enter student name: Mani
Enter student name: Mani
Enter student age: 18
Enter marks for each subject (press Enter with empty input to finish):
Enter mark for subject 1 (or press Enter to finish): 95
Enter mark for subject 1 (or press Enter to finish): 95
Enter mark for subject 2 (or press Enter to finish): 96
Enter mark for subject 3 (or press Enter to finish): 97
Enter mark for subject 3 (or press Enter to finish): 97
Enter mark for subject 4 (or press Enter to finish):
Enter mark for subject 4 (or press Enter to finish):

=====
Student Details:
Name: Mani
Age: 18
Marks: 95.0, 96.0, 97.0
Total Marks: 288.0
Average Marks: 96.00
=====
PS C:\Users\nadhi\OneDrive\Desktop\AI> |
```

Observation:

Excellent readability & modularity – clear class structure, meaningful names, and docstrings.

User input validation – prevents empty names, invalid ages, and out-of-range marks.

Flexible marks storage – uses a list and `sum(self.marks)` for totals.

Extra feature: `average()` method makes it more useful.

Main function is clean and follows `if __name__ == "__main__":` best practice.

Task – 04

Refactor this inefficient loop with AI help.

Prompt:

Refactor the following Python code to make it more **concise** and **Pythonic**:

- Replace the inefficient for loop with a **list comprehension**.
- Allow the user to **input numbers** (comma-separated) at runtime.
- Output the original list and the list of squares.

```
nums = [1,2,3,4,5,6,7,8,9,10]
```

```
squares = []
```

```
for i in nums:
```

```
    squares.append(i * i)
```

Code generated:

```
squares_calculator.py > main
1  #!/usr/bin/env python3
2  """
3  Squares Calculator - A Pythonic approach to calculating squares of numbers.
4
5  This script demonstrates:
6  - List comprehensions for concise code
7  - User input handling with validation
8  - Clean output formatting
9  """
10
11 def get_numbers_from_user():
12     """Get comma-separated numbers from user input."""
13     while True:
14         try:
15             user_input = input("Enter numbers separated by commas: ")
16             # Split by comma, strip whitespace, and convert to integers
17             numbers = [int(num.strip()) for num in user_input.split(',')]
18             return numbers
19         except ValueError:
20             print("Invalid input! Please enter numbers separated by commas (e.g., 1,2,3,4,5)")
21         except KeyboardInterrupt:
22             print("\nExiting...")
23             exit(0)
24
25
26 def calculate_squares(numbers):
27     """Calculate squares using list comprehension."""
28     return [num * num for num in numbers]
29
30
31 def main():
32     """Main function to run the squares calculator."""
33     print("Squares Calculator")
34     print("=" * 20)
35
36     # Get numbers from user
37     nums = get_numbers_from_user()
38
39     # Calculate squares using list comprehension (replaces the inefficient for loop)
40     squares = calculate_squares(nums)
```

```
refactored_student.py  squares_calculator.py X
squares_calculator.py > main
31 def main():
40     squares = calculate_squares(nums)
41
42     # Display results
43     print(f"\nOriginal numbers: {nums}")
44     print(f"Squares:           {squares}")
45
46     # Optional: Show the transformation in a more detailed format
47     print("\nDetailed view:")
48     for num, square in zip(nums, squares):
49         print(f"{num}^2 = {square}")
50
51
52 if __name__ == "__main__":
53     main()
54
```

Output:

```

=====
PS C:\Users\nadhi\OneDrive\Desktop\AI> & C:/Users/nadhi/anaconda3/python.exe c:/Users/nadhi/OneDrive/Desktop/AI/squares_calculator.py
Squares Calculator
=====
Enter numbers separated by commas: 2,3

Original numbers: [2, 3]
Squares:          [4, 9]

Detailed view:
22 = 4
32 = 9

Original numbers: [2, 3]
Squares:          [4, 9]

Detailed view:
22 = 4
32 = 9
22 = 4
32 = 9
PS C:\Users\nadhi\OneDrive\Desktop\AI> 

```

Observation:

Excellent use of list comprehension – concise and Pythonic.

Robust input handling – strips whitespace, validates integers, handles KeyboardInterrupt.

Well-structured code – separate functions for input, calculation, and main logic.

Clear and clean output – shows original numbers, squares, and a detailed mapping.

Good documentation – docstrings and module description included.