

AI-Assignment-10.1

NAME:Karthikeya

HT.NO:2303A51306

Batch:05

Task Description #1 – Syntax and Logic Errors

Prompt:

Task: Use AI to identify and fix syntax and logic errors in a faulty [Python script].

Sample Input Code:

Calculate average score of a student

```
def calc_average(marks):  
    total = 0  
    for m in marks:  
        total += m  
    average = total / len(marks)  
    return avrage # Typo here  
marks = [85, 90, 78, 92]  
print("Average Score is ", calc_average(marks))
```

Expected Output:

- Corrected and runnable Python code with explanations of the fixes.

Refactored Code:

```
def calc_average(marks):  
    total = 0  
    for m in marks:  
        total += m  
    average = total / len(marks)  
    return average  
marks = [85, 90, 78, 92]  
print("Average Score is ", calc_average(marks))
```

Output:

Average Score is 86.25

```
17 # Explanation of the debugging steps taken to identify and fix the issues.
18 # The original code has several issues:
19 # 1. The function definition is not properly indented.
20 # 2. The variable 'average' is misspelled as 'avrage' in the return statement.
21 # 3. The print statement is missing a closing parenthesis.
22 # Here is the corrected code with explanations of the fixes:
23
24 def calc_average(marks):
25     total = 0
26     for i in marks:
27         total += i # This line is now properly indented
28     average = total / len(marks) # The variable 'average' is correctly spelled
29     return average # Return the correct variable
30
31 marks = [85, 90, 78, 92]
32 print("Average score is ", calc_average(marks)) # Added the missing closing parenthesis
33
34 # Explanation of the fixes:
35 # - Proper indentation is crucial in Python as it defines the scope of loops and functions.
36 # - The original code's lack of indentation would lead to a syntax error.
37 # - The misspelled variable 'avrage' would cause a NameError when the function tries to return it, as it is not defined.
38 # - Correcting it to 'average' ensures that the function returns the intended value.
39 # - The missing closing parenthesis in the print statement would also lead to a syntax error.
40 # - By adding it, we ensure that the print function is called correctly, allowing the program to run without errors.
41 # Overall, these fixes improve the robustness of the code by ensuring it is syntactically correct and functions as intended.
```

```
PS C:\Users\GAJE HARIKA\OneDrive\Desktop\AI Coding> & "C:/Users/GAJE HARIKA/AppData/Local/Python/pythoncore-3.14-64/python.exe"
" C:/Users/GAJE HARIKA/OneDrive/Desktop/AI Coding/lab_10.1.py"
Average Score is : 86.25
PS C:\Users\GAJE HARIKA\OneDrive\Desktop\AI Coding>
```

Explanation of the fixes:

- Proper indentation is crucial in Python as it defines the scope of loops and functions. The original code's lack of indentation would lead to a syntax error.
- The misspelled variable 'avrage' would cause a NameError when the function tries to return it, as it is not defined. Correcting it to 'average' ensures that the function returns the intended value.
- The missing closing parenthesis in the print statement would also lead to a syntax error. By adding it, we ensure that the print function is called correctly, allowing the program to run without errors. Overall, these fixes improve the robustness of the code by ensuring it is syntactically correct and functions as intended.

Task Description #2 – PEP 8 Compliance

Prompt:

Task: Use AI to refactor Python code to follow PEP 8 style guidelines.

Sample Input Code:

```
def area_of_rect(L,B): return L*B
print(area_of_rect(10,20))
```

Expected Output:

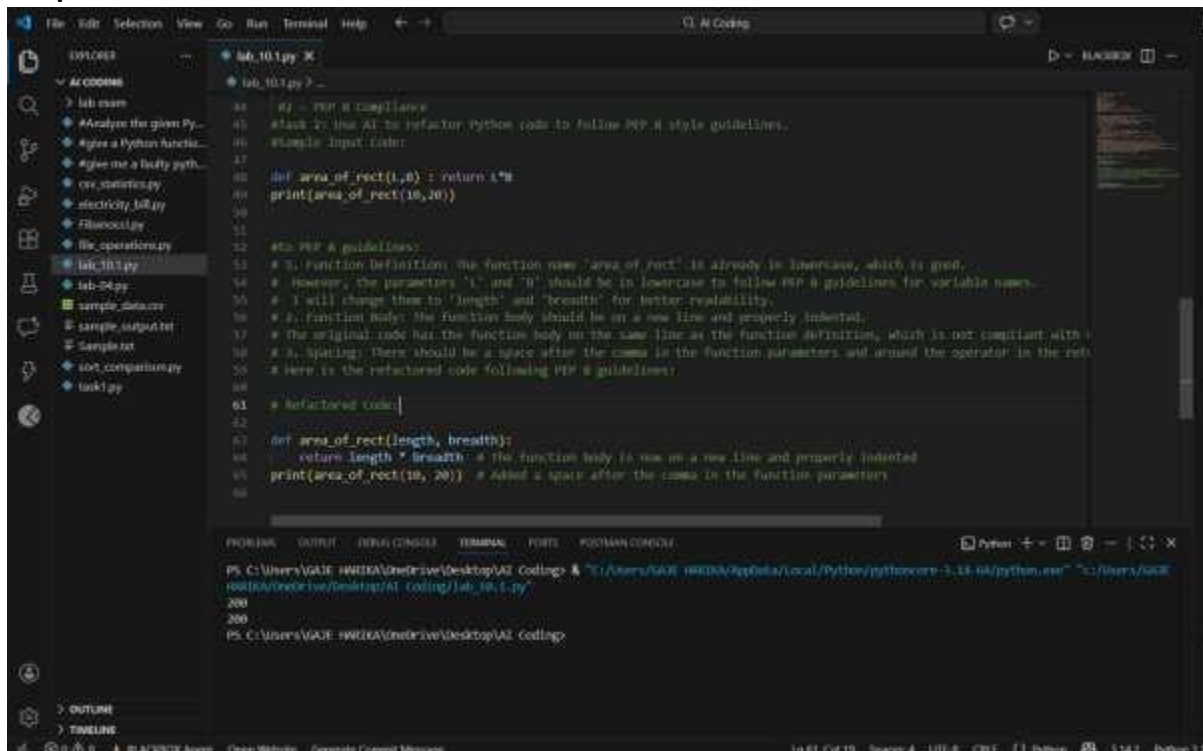
- Well-formatted PEP 8-compliant Python code.

Refactored Code:

```
def area_of_rect(length, breadth):
```

```
    return length * breadth
print(area_of_rect(10, 20))
```

Output:



```
44 # AI - PEP 8 Compliance
45 Ask AI to refactor Python code to follow PEP 8 style guidelines.
46 Sample Input Code:
47
48 def area_of_rect(l,b): return l*b
49 print(area_of_rect(10,20))
50
51
52 # PEP 8 guidelines:
53 # 1. Function Definition: The function name 'area_of_rect' is already in lowercase, which is good.
54 # However, the parameters 'l' and 'b' should be in lowercase to follow PEP 8 guidelines for variable names.
55 # I will change them to 'length' and 'breadth' for better readability.
56 # 2. Function Body: The function body should be on a new line and properly indented.
57 # The original code has the function body on the same line as the function definition, which is not compliant with PEP 8.
58 # 3. Spacing: There should be a space after the comma in the function parameters and around the operator in the return statement.
59 # Here is the refactored code following PEP 8 guidelines:
60
61 # Refactored Code:
62
63 def area_of_rect(length, breadth):
64     return length * breadth # The function body is now on a new line and properly indented
65 print(area_of_rect(10, 20)) # Added a space after the comma in the function parameters
66
```

Explanation of the fixes:

1. Function Definition: The function name 'area_of_rect' is already in lowercase, which is good. However, the parameters 'L' and 'B' should be in lowercase to follow PEP 8 guidelines for variable names. I will change them to 'length' and 'breadth' for better readability.
2. Function Body: The function body should be on a new line and properly indented. The original code has the function body on the same line as the function definition, which is not compliant with PEP 8.
3. Spacing: There should be a space after the comma in the function parameters and around the operator in the return statement for better readability.

Task Description

#3 – Readability Enhancement

Prompt:

Task: Use AI to make code more readable without changing its logic.

Sample Input Code:

```
def c(x,y):
return x*y/100
a=200
b=15
print(c(a,b))
```

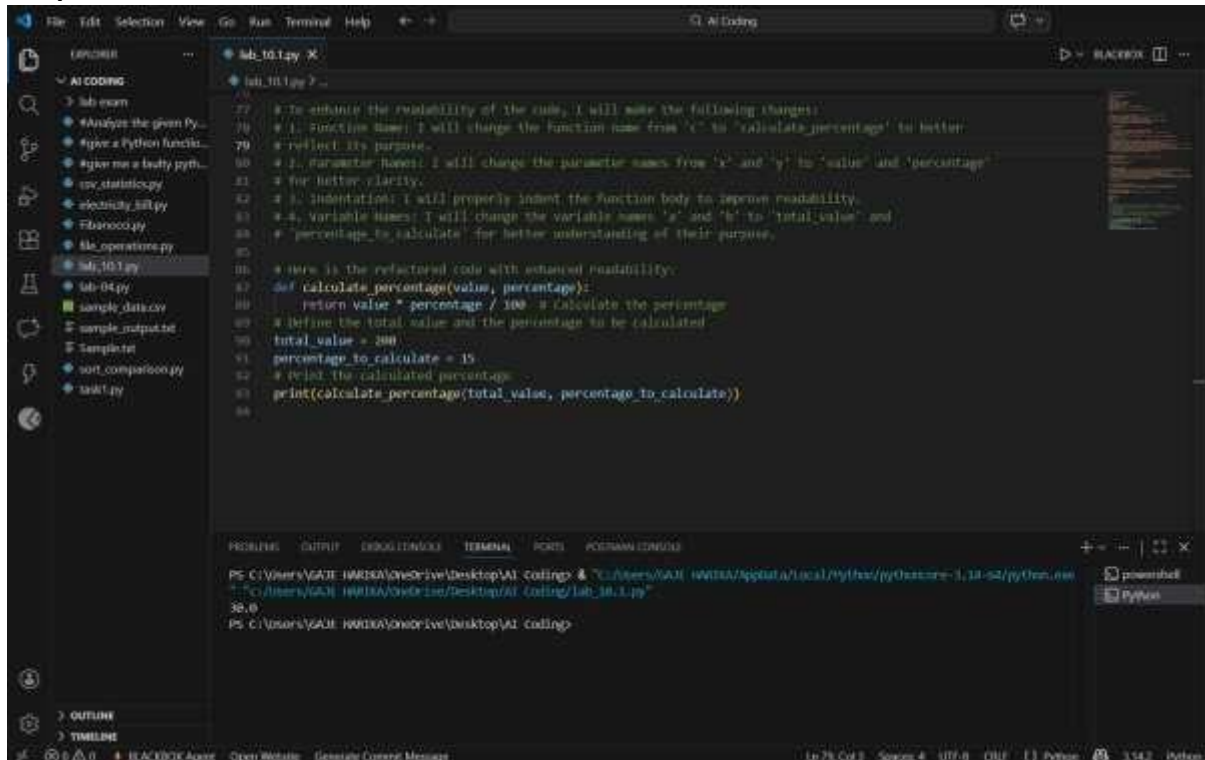
Expected Output:

- Python code with descriptive variable names, inline comments, and clear formatting.

Refactored Code:

```
def calculate_percentage(value, percentage):  
    return value * percentage / 100  
  
total_value = 200  
percentage_to_calculate = 15  
# Print the calculated percentage  
print(calculate_percentage(total_value, percentage_to_calculate))
```

Output:



Explanation of the fixes:

1. Function Name: I will change the function name from 'c' to 'calculate_percentage' to better reflect its purpose.
2. Parameter Names: I will change the parameter names from 'x' and 'y' to 'value' and 'percentage' for better clarity.
3. Indentation: I will properly indent the function body to improve readability.
4. Variable Names: I will change the variable names 'a' and 'b' to 'total_value' and 'percentage_to_calculate' for better understanding of their purpose.

Task Description #4 – Refactoring for Maintainability

Prompt:

Task: Use AI to break repetitive or long code into reusable functions.

Sample Input Code:

```
students = ["Alice", "Bob", "Charlie"]  
print("Welcome", students[0])  
print("Welcome", students[1])  
print("Welcome", students[2])
```

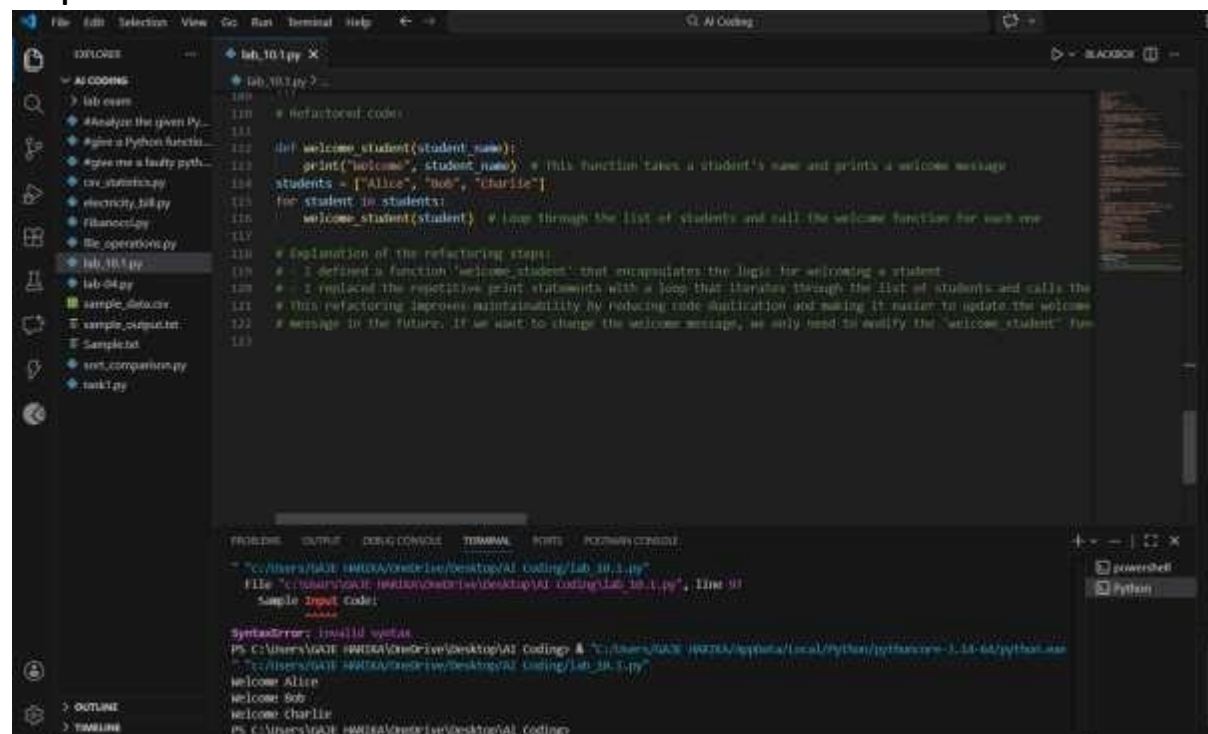
Expected Output:

- Modular code with reusable functions.

Refactored Code:

```
def welcome_student(student_name):  
    print("Welcome", student_name)  
students = ["Alice", "Bob", "Charlie"]  
for student in students:  
    welcome_student(student)
```

Output:



The screenshot shows a VS Code editor with a file named 'lab_10.1.py'. The code defines a function 'welcome_student' that takes a student name and prints a welcome message. It then creates a list of students ['Alice', 'Bob', 'Charlie'] and uses a for loop to call the 'welcome_student' function for each student. The terminal output shows the execution of the code, which prints 'Welcome Alice', 'Welcome Bob', and 'Welcome Charlie' on separate lines. The terminal also shows the command 'python lab_10.1.py' and the output of the program.

Explanation of the fixes:

- I defined a function 'welcome_student' that encapsulates the logic for welcoming a student
- I replaced the repetitive print statements with a loop that iterates through the list of students and calls the 'welcome_student' function for each student. This refactoring improves maintainability by reducing code duplication and making it easier to update the welcome message in the future. If we want to change the welcome message, we only need to modify the 'welcome_student' function, and it will automatically apply to all students without needing to change multiple lines of code.

Task Description #5 – Performance Optimization

Prompt:

Task: Use AI to make the code run faster.

Sample Input Code:

```
# Find squares of numbers  
nums = [i for i in range(1,1000000)]
```

```
squares = []
for n in nums:
    squares.append(n**2)
print(len(squares))
```

Expected Output:

- Optimized code using list comprehensions or vectorized operations.

Refactored Code:

```
nums = [i for i in range(1, 1000000)]
squares = [n**2 for n in nums]
print(len(squares))
```

Output:

The screenshot shows a code editor with a file explorer on the left containing files like 'lab_10_1.py', 'sample_data.csv', and 'sample_output.txt'. The main editor displays the refactored Python code for calculating squares. Below the code, the 'TERMINAL' tab shows the execution output, which includes a 'SyntaxError: invalid syntax' message and a list of names: Alice, Bob, Charlie. The status bar at the bottom indicates the file is 'lab_10_1.py' and the interpreter is 'Python 3.11.2'.

Explanation of the fixes:

- The list comprehension allows us to create the list of squares in a single line of code, which is more efficient than using a for loop with append.
- List comprehensions are optimized in Python and can be significantly faster than traditional loops, especially when dealing with large datasets.
- By using a list comprehension, we reduce the overhead of multiple append operations and improve the overall performance of the code, making it run faster when calculating the squares of a large list of numbers.

Task Description #6 – Complexity Reduction

Prompt:

Task: Use AI to simplify overly complex logic.

Sample Input Code:

```
def grade(score):  
    if score >= 90:  
        return "A"  
    else:  
        if score >= 80:  
            return "B"  
        else:  
            if score >= 70:  
                return "C"  
            else:  
                if score >= 60:  
                    return "D"  
                else:  
                    return "F"
```

Expected Output:

- Cleaner logic using elif or dictionary mapping.

Refactored Code:

```
def grade(score):  
    if score >= 90:  
        return "A"  
    elif score >= 80:  
        return "B"  
    elif score >= 70:  
        return "C"  
    elif score >= 60:  
        return "D"  
    else:  
        return "F"
```

Output:


```
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
```

Explanation of the fixes:

- By using elif statements, we have reduced the number of nested blocks, which makes the code easier to read and understand.
- The original code's nested structure can be confusing and harder to maintain, especially if we need to add more grade categories in the future. The refactored code is more straightforward and allows for easier modifications if needed.