

ASSIGNMENT 10.4

D.Charmi

2303A51314

Batch – 05

TASK – 01:

Wrong Code :



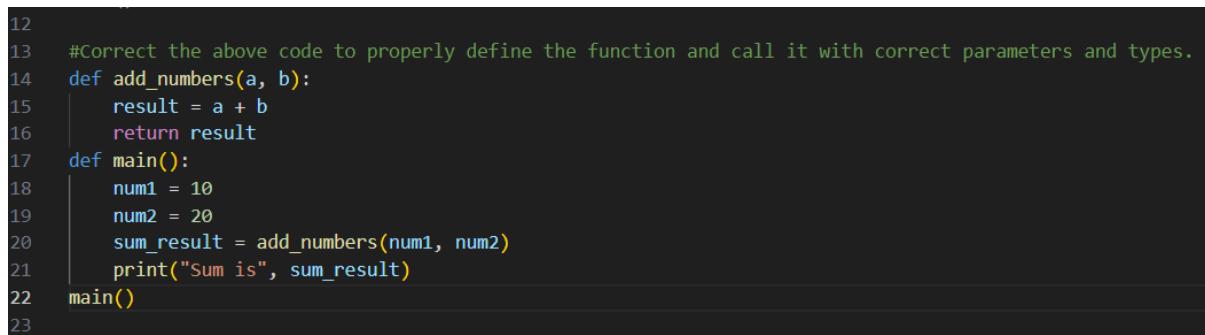
```
10.4.py > ...
1  def addNumbers(a b)
2  result = a + b
3  return result
4
5  def main():
6  num1 = 10
7  num2 = "20"
8  sum = addNumbers(num1, num2)
9  print("Sum is", sum)
10
11 main()
```

Output :



```
PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding> & c:/Users/devis/AppData/Local/Python/pythoncore-3.14-64/python.exe "c:/Users/devis/OneDrive/Desktop/AI Assisted Coding\10.4.py"
File "c:/Users/devis/OneDrive/Desktop/AI Assisted Coding\10.4.py", line 1
  def addNumbers(a b)
          ^
SyntaxError: invalid syntax
PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding>
```

Corrected Code:



```
12
13  #Correct the above code to properly define the function and call it with correct parameters and types.
14  def add_numbers(a, b):
15      result = a + b
16      return result
17  def main():
18      num1 = 10
19      num2 = 20
20      sum_result = add_numbers(num1, num2)
21      print("Sum is", sum_result)
22  main()
23
```

Output :



```
PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding> & c:/Users/devis/AppData/Local/Python/pythoncore-3.14-64/python.exe "c:/Users/devis/OneDrive/Desktop/AI Assisted Coding\10.4.py"
Sum is 30
PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding>
```

Explanation :

The given program has syntax errors like missing commas and wrong indentation.
The function name is written differently while calling it, causing an error.
A string value is added to an integer, which leads to a type error.
Overall, the code cannot run because of these basic mistakes.

All syntax and indentation errors are fixed.
Function and variable names are corrected and standardized.
Both values are integers, so addition works correctly.
The program now runs successfully and prints the correct sum.

TASK – 02:

Inefficient Code:

```
PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding> ...  
1 def find_duplicates(data):  
2     duplicates = []  
3     for i in range(len(data)):  
4         for j in range(i + 1, len(data)):  
5             if data[i] == data[j] and data[i] not in duplicates:  
6                 duplicates.append(data[i])  
7     return duplicates  
8 # Example usage:  
9 data = [1, 2, 3, 4, 2, 5, 1]  
10 duplicates = find_duplicates(data)  
11 print("Duplicate elements:", duplicates)  
12
```

Output :

```
PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding> & C:/Users/devis/AppData/Local/Python/pythoncore-3.14-64/python.exe "c:/Users/devis/OneDrive/Desktop/AI Assisted Coding/10.4.py"  
Duplicate elements: [1, 2]  
PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding>
```

Improved version of Code :

```
12 #Improve the function by using a set to track seen elements and duplicates, which will reduce the time complexity to O(n):  
13 def find_duplicates(data):  
14     seen = set()  
15     duplicates = set()  
16     for item in data:  
17         if item in seen:  
18             duplicates.add(item)  
19         else:  
20             seen.add(item)  
21     return list(duplicates)  
22 # Example usage:  
23 data = [1, 2, 3, 4, 2, 5, 1]  
24 duplicates = find_duplicates(data)  
25 print("Duplicate elements:", duplicates)  
26 ...  
27 This function takes a list of data as input and returns a list of duplicate elements found in the input list.  
28  
29 :param data: List of elements to check for duplicates  
30 :return: A list of duplicate elements  
31 ...  
32
```

Output :

Explanation :

- The original function uses nested loops, which compare each element with every other element, resulting in $O(n^2)$ time complexity.
 - This approach becomes slow and inefficient when the input list is large.
 - The improved function uses sets to track seen elements and duplicates.
 - Set lookups are fast ($O(1)$), so each element is checked only once.
 - This reduces the overall time complexity to $O(n)$.
 - The optimized function produces the same correct output but with much better performance and readability.

TASK – 03:

Poorly Structured Code :

```
4 10.4.py > ...
5
6 1   def f(a):
7     x=0
8     for i in a:
9       if i%2==0:
10         x=x+i
11
12 12   return x
13
14 13
15 14
16 15
17 16
18 17
19 18
20 19
21 20
22 21
23 22
24 23
25 24
26 25
27 26
28 27
29 28
30 29
31 30
32 31
33 32
34 33
35 34
36 35
37 36
38 37
39 38
40 39
41 40
42 41
43 42
44 43
45 44
46 45
47 46
48 47
49 48
50 49
51 50
52 51
53 52
54 53
55 54
56 55
57 56
58 57
59 58
60 59
61 60
62 61
63 62
64 63
65 64
66 65
67 66
68 67
69 68
70 69
71 70
72 71
73 72
74 73
75 74
76 75
77 76
78 77
79 78
80 79
81 80
82 81
83 82
84 83
85 84
86 85
87 86
88 87
89 88
90 89
91 90
92 91
93 92
94 93
95 94
96 95
97 96
98 97
99 98
100 99
101 100
102 101
103 102
104 103
105 104
106 105
107 106
108 107
109 108
110 109
111 110
112 111
113 112
114 113
115 114
116 115
117 116
118 117
119 118
120 119
121 120
122 121
123 122
124 123
125 124
126 125
127 126
128 127
129 128
130 129
131 130
132 131
133 132
134 133
135 134
136 135
137 136
138 137
139 138
140 139
141 140
142 141
143 142
144 143
145 144
146 145
147 146
148 147
149 148
150 149
151 150
152 151
153 152
154 153
155 154
156 155
157 156
158 157
159 158
160 159
161 160
162 161
163 162
164 163
165 164
166 165
167 166
168 167
169 168
170 169
171 170
172 171
173 172
174 173
175 174
176 175
177 176
178 177
179 178
180 179
181 180
182 181
183 182
184 183
185 184
186 185
187 186
188 187
189 188
190 189
191 190
192 191
193 192
194 193
195 194
196 195
197 196
198 197
199 198
200 199
201 200
202 201
203 202
204 203
205 204
206 205
207 206
208 207
209 208
210 209
211 210
212 211
213 212
214 213
215 214
216 215
217 216
218 217
219 218
220 219
221 220
222 221
223 222
224 223
225 224
226 225
227 226
228 227
229 228
230 229
231 230
232 231
233 232
234 233
235 234
236 235
237 236
238 237
239 238
240 239
241 240
242 241
243 242
244 243
245 244
246 245
247 246
248 247
249 248
250 249
251 250
252 251
253 252
254 253
255 254
256 255
257 256
258 257
259 258
260 259
261 260
262 261
263 262
264 263
265 264
266 265
267 266
268 267
269 268
270 269
271 270
272 271
273 272
274 273
275 274
276 275
277 276
278 277
279 278
280 279
281 280
282 281
283 282
284 283
285 284
286 285
287 286
288 287
289 288
290 289
291 290
292 291
293 292
294 293
295 294
296 295
297 296
298 297
299 298
300 299
301 300
302 301
303 302
304 303
305 304
306 305
307 306
308 307
309 308
310 309
311 310
312 311
313 312
314 313
315 314
316 315
317 316
318 317
319 318
320 319
321 320
322 321
323 322
324 323
325 324
326 325
327 326
328 327
329 328
330 329
331 330
332 331
333 332
334 333
335 334
336 335
337 336
338 337
339 338
340 339
341 340
342 341
343 342
344 343
345 344
346 345
347 346
348 347
349 348
350 349
351 350
352 351
353 352
354 353
355 354
356 355
357 356
358 357
359 358
360 359
361 360
362 361
363 362
364 363
365 364
366 365
367 366
368 367
369 368
370 369
371 370
372 371
373 372
374 373
375 374
376 375
377 376
378 377
379 378
380 379
381 380
382 381
383 382
384 383
385 384
386 385
387 386
388 387
389 388
390 389
391 390
392 391
393 392
394 393
395 394
396 395
397 396
398 397
399 398
400 399
401 400
402 401
403 402
404 403
405 404
406 405
407 406
408 407
409 408
410 409
411 410
412 411
413 412
414 413
415 414
416 415
417 416
418 417
419 418
420 419
421 420
422 421
423 422
424 423
425 424
426 425
427 426
428 427
429 428
430 429
431 430
432 431
433 432
434 433
435 434
436 435
437 436
438 437
439 438
440 439
441 440
442 441
443 442
444 443
445 444
446 445
447 446
448 447
449 448
450 449
451 450
452 451
453 452
454 453
455 454
456 455
457 456
458 457
459 458
460 459
461 460
462 461
463 462
464 463
465 464
466 465
467 466
468 467
469 468
470 469
471 470
472 471
473 472
474 473
475 474
476 475
477 476
478 477
479 478
480 479
481 480
482 481
483 482
484 483
485 484
486 485
487 486
488 487
489 488
490 489
491 490
492 491
493 492
494 493
495 494
496 495
497 496
498 497
499 498
500 499
501 500
502 501
503 502
504 503
505 504
506 505
507 506
508 507
509 508
510 509
511 510
512 511
513 512
514 513
515 514
516 515
517 516
518 517
519 518
520 519
521 520
522 521
523 522
524 523
525 524
526 525
527 526
528 527
529 528
530 529
531 530
532 531
533 532
534 533
535 534
536 535
537 536
538 537
539 538
540 539
541 540
542 541
543 542
544 543
545 544
546 545
547 546
548 547
549 548
550 549
551 550
552 551
553 552
554 553
555 554
556 555
557 556
558 557
559 558
560 559
561 560
562 561
563 562
564 563
565 564
566 565
567 566
568 567
569 568
570 569
571 570
572 571
573 572
574 573
575 574
576 575
577 576
578 577
579 578
580 579
581 580
582 581
583 582
584 583
585 584
586 585
587 586
588 587
589 588
590 589
591 590
592 591
593 592
594 593
595 594
596 595
597 596
598 597
599 598
600 599
601 600
602 601
603 602
604 603
605 604
606 605
607 606
608 607
609 608
610 609
611 610
612 611
613 612
614 613
615 614
616 615
617 616
618 617
619 618
620 619
621 620
622 621
623 622
624 623
625 624
626 625
627 626
628 627
629 628
630 629
631 630
632 631
633 632
634 633
635 634
636 635
637 636
638 637
639 638
640 639
641 640
642 641
643 642
644 643
645 644
646 645
647 646
648 647
649 648
650 649
651 650
652 651
653 652
654 653
655 654
656 655
657 656
658 657
659 658
660 659
661 660
662 661
663 662
664 663
665 664
666 665
667 666
668 667
669 668
670 669
671 670
672 671
673 672
674 673
675 674
676 675
677 676
678 677
679 678
680 679
681 680
682 681
683 682
684 683
685 684
686 685
687 686
688 687
689 688
690 689
691 690
692 691
693 692
694 693
695 694
696 695
697 696
698 697
699 698
700 699
701 700
702 701
703 702
704 703
705 704
706 705
707 706
708 707
709 708
710 709
711 710
712 711
713 712
714 713
715 714
716 715
717 716
718 717
719 718
720 719
721 720
722 721
723 722
724 723
725 724
726 725
727 726
728 727
729 728
730 729
731 730
732 731
733 732
734 733
735 734
736 735
737 736
738 737
739 738
740 739
741 740
742 741
743 742
744 743
745 744
746 745
747 746
748 747
749 748
750 749
751 750
752 751
753 752
754 753
755 754
756 755
757 756
758 757
759 758
760 759
761 760
762 761
763 762
764 763
765 764
766 765
767 766
768 767
769 768
770 769
771 770
772 771
773 772
774 773
775 774
776 775
777 776
778 777
779 778
780 779
781 780
782 781
783 782
784 783
785 784
786 785
787 786
788 787
789 788
790 789
791 790
792 791
793 792
794 793
795 794
796 795
797 796
798 797
799 798
800 799
801 800
802 801
803 802
804 803
805 804
806 805
807 806
808 807
809 808
810 809
811 810
812 811
813 812
814 813
815 814
816 815
817 816
818 817
819 818
820 819
821 820
822 821
823 822
824 823
825 824
826 825
827 826
828 827
829 828
830 829
831 830
832 831
833 832
834 833
835 834
836 835
837 836
838 837
839 838
840 839
841 840
842 841
843 842
844 843
845 844
846 845
847 846
848 847
849 848
850 849
851 850
852 851
853 852
854 853
855 854
856 855
857 856
858 857
859 858
860 859
861 860
862 861
863 862
864 863
865 864
866 865
867 866
868 867
869 868
870 869
871 870
872 871
873 872
874 873
875 874
876 875
877 876
878 877
879 878
880 879
881 880
882 881
883 882
884 883
885 884
886 885
887 886
888 887
889 888
890 889
891 890
892 891
893 892
894 893
895 894
896 895
897 896
898 897
899 898
900 899
901 900
902 901
903 902
904 903
905 904
906 905
907 906
908 907
909 908
910 909
911 910
912 911
913 912
914 913
915 914
916 915
917 916
918 917
919 918
920 919
921 920
922 921
923 922
924 923
925 924
926 925
927 926
928 927
929 928
930 929
931 930
932 931
933 932
934 933
935 934
936 935
937 936
938 937
939 938
940 939
941 940
942 941
943 942
944 943
945 944
946 945
947 946
948 947
949 948
950 949
951 950
952 951
953 952
954 953
955 954
956 955
957 956
958 957
959 958
960 959
961 960
962 961
963 962
964 963
965 964
966 965
967 966
968 967
969 968
970 969
971 970
972 971
973 972
974 973
975 974
976 975
977 976
978 977
979 978
980 979
981 980
982 981
983 982
984 983
985 984
986 985
987 986
988 987
989 988
990 989
991 990
992 991
993 992
994 993
995 994
996 995
997 996
998 997
999 998
1000 999
```

Output :

```
4.py
PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding> & C:/Users/devis/AppData/Local/Python/pythoncore-3.14-64/python.exe "c:/Users/devis/OneDrive/Desktop/AI Assisted Coding/10.
4.py"
12
PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding> []
```

Refactored Code :

```
11  #Refactor the above code with proper naming conventions
12  def sum_of_even_numbers(numbers):
13      ...
14
15      This function takes a list of numbers as input and returns the sum of even numbers.
16
17      :param numbers: List of integers
18      :return: Sum of even numbers
19      ...
20
21      even_sum = 0
22      for number in numbers:
23          if number % 2 == 0:
24              even_sum += number
25      return even_sum
26
27  # Example usage:
28  numbers = [1, 2, 3, 4, 5, 6]
29  print(sum_of_even_numbers(numbers))
30
```

Output :

```
● PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding> & C:/Users/d
4.py"
12
○ PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding>
```

Explanation :

- The unclear function name f was renamed to sum_of_even_numbers to clearly describe its purpose.
- Variable names like a, x, and i were replaced with meaningful names (numbers, even_sum, number).
- Proper indentation and formatting were applied for better readability.
- A docstring was added to explain the function's purpose, parameters, and return value.
- The logic remains the same, but the code is now easy to read and maintain.

TASK – 04

Insecure Code :

```
10.4.py > ...
1  import sqlite3
2  conn = sqlite3.connect("shop.db")
3  cursor = conn.cursor()
4  cursor.execute("""
5    CREATE TABLE IF NOT EXISTS products (
6      id INTEGER PRIMARY KEY AUTOINCREMENT,
7      name TEXT NOT NULL,
8      price REAL
9    )
10   """)
11  cursor.execute("INSERT INTO products (name, price) VALUES (?, ?)",
12    ("Laptop", 75000))
13  cursor.execute("INSERT INTO products (name, price) VALUES (?, ?)",
14    ("Phone", 35000))
15  conn.commit()
16  conn.close()
17  print("Products table created successfully!")
18  def get_product(product_name):
19    conn = sqlite3.connect("shop.db")
20    cursor = conn.cursor()
21    query = "SELECT * FROM products WHERE name = '" + product_name + "'"
22    cursor.execute(query)
23    result = cursor.fetchall()
24    conn.close()
25    return result
26  # Example usage
27  name = input("Enter product name: ")
28  print(get_product(name))
29  |
```

Output :

```
PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding> & C:/users/devis/10.4.py"
Products table created successfully!
> Enter product name: Laptop
[(1, 'Laptop', 75000.0), (3, 'Laptop', 75000.0)]
PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding> []
```

Secured Code:

```
#Secure version using parameterized queries
def get_product_secure(product_name):
    conn = sqlite3.connect("shop.db")
    cursor = conn.cursor()
    query = "SELECT * FROM products WHERE name = ?"
    cursor.execute(query, (product_name,))
    result = cursor.fetchall()
    conn.close()
    return result

# Example usage
name = input("Enter product name: ")
print(get_product_secure(name))
```

Output :

```
PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding> & C:/Users/devis/!4.py"
Products table created successfully!
> Enter product name: Laptop
[(1, 'Laptop', 75000.0), (3, 'Laptop', 75000.0)]
PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding> []
```

Explanation :

- The code contains two different versions of the same functionality.
- The first version is an insecure example.
- It uses string concatenation to build SQL queries, which is unsafe and vulnerable to SQL injection.
- The second version is the secure implementation that actually runs.
- It uses parameterized queries (?), which safely handle user input.
- The secure version is safe, reliable, and suitable for production use.

TASK – 05:

Poor Code:

```
 10.4.py > ...
1  def p(a,b):
2      r=0
3      for i in a:
4          if i==b:
5              r=r+1
6      return r
7      ''' Docstring for p
8
9      This function counts the occurrences of a specific element in a list.
10
11     :param a: List of elements
12     :param b: Element to count
13     :return: Count of occurrences of b in a
14     '''
15
16     # Example usage:
17     a=[1,2,3,4,1,2,1]
18     b=1
19     count=p(a,b)
20     print("Count of", b, "in list:", count)
21     print(p.__doc__)
```

Output :

```
PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding> & C:/Users/devis/OneDrive/Desktop/AI Assisted Coding> python 10.4.py
Count of 1 in list: 3
None
PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding> []
```

Improved version :

```
21
22     #Improve the above code in terms of readability and documentation.
23     def count_occurrences(elements, target):
24         """
25             This function counts the occurrences of a specific element in a list.
26
27             :param elements: List of elements
28             :param target: Element to count
29             :return: Count of occurrences of target in elements
30             """
31
32         count = 0
33         for element in elements:
34             if element == target:
35                 count += 1
36         return count
37
38     # Example usage:
39     elements = [1, 2, 3, 4, 1, 2, 1]
40     target = 1
41     occurrences = count_occurrences(elements, target)
42     print("Count of", target, "in list:", occurrences)
43     print(count_occurrences.__doc__)
44
```

Output :

- PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding> & C:/Users/devis/4.py"
Count of 1 in list: 3

This function counts the occurrences of a specific element in a list.

:param elements: List of elements
:param target: Element to count
:return: Count of occurrences of target in elements
- PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding> □

Explanation :

- The unclear function name p was renamed to count_occurrences to describe its purpose clearly.
- Variable names a and b were replaced with meaningful names (elements, target).
- Proper indentation and formatting were applied for better readability.
- A clear and correct docstring was added inside the function.
- The code logic remains the same, but it is now easy to understand and maintain.