

Assignment– 5.5

2303A51655

Batch:-29

Task1:TransparencyinAlgorithmOptimization

PROMPT:

GeneratePythoncodefortwoprime-checkingmethodsandexplainhowthe optimized version improves performance.give two codes one for naive approach and optimized approach

CODEandOUTPUT:

```
AI-5.5.py
1 #Generate Python code for two prime-checking methods and explain how the optimized version improves performance.give two codes one for naive approach and optim
2 # Naive Approach to Check for Prime Number
3 def is_prime_naive(n):
4     if n <= 1:
5         return False
6     for i in range(2, n):
7         if n % i == 0:
8             return False
9     return True
10 # Optimized Approach to Check for Prime Number
11 def is_prime_optimized(n):
12     if n <= 1:
13         return False
14     if n <= 3:
15         return True
16     if n % 2 == 0 or n % 3 == 0:
17         return False
18     i = 5
19     while i * i <= n:
20         if n % i == 0 or n % (i + 2) == 0:
21             return False
22         i += 6
23     return True
24 # Explanation of Optimization:
25 # 1. The naive approach checks for factors from 2 to n-1, which results in O(N) time complexity.
26 # 2. The optimized approach reduces the number of checks significantly.
27 # 3. It first eliminates numbers less than or equal to 1, and directly returns
28 # 4. True for 2 and 3.
29 # 5. It eliminates even numbers and multiples of 3 right away.
30 # 6. It then checks for factors only up to the square root of n, since if n is divisible
31 # 7. by any number greater than its square root, it must also be divisible by a number
32 # 8. smaller than its square root.
33 # 9. It checks only numbers of the form 6k ± 1, as all prime numbers greater than 3
34 # 5 can be expressed in this form, further reducing the number of checks.
35 # This results in an average time complexity of O(√N), making it much more efficient for larger numbers.
36 # Example Usage
37 number = 29
38 print("Naive: Is (number) prime? (is_prime_naive(number))")
39 print("Optimized: Is (number) prime? (is_prime_optimized(number))")
40
```

```
56856' '--' 'D:\AI\AI-5.5.py'
Naive: Is 29 prime? True
Optimized: Is 29 prime? True
PS D:\AI>
```

Justification:

We used two methods to check prime numbers to understand algorithm transparency and optimization.

- The naive approach checks all numbers from 2 to $n-1$, which works but is slow for large numbers.
- The optimized approach checks only up to the square root of the number, which reduces unnecessary calculations.

This shows how AI-generated code should be understood and improved, not blindly used. Optimized algorithms save time and computing resources, which is an ethical responsibility of developers.

Task 2: Transparency in Recursive Algorithms

PROMPT:

Generate a recursive function to calculate fibonacci numbers and add comments explaining recursion and base case and recursive case.

CODE and OUTPUT:

```
#Generate a recursive function to calculate fibonacci numbers and add comments explaining recursion and base case and recursive case.
def fibonacci(n):
    # Base Case: If n is 0 or 1, return n (the first two fibonacci numbers)
    if n <= 1:
        return n
    # Recursive Case: Return the sum of the two preceding Fibonacci numbers
    return fibonacci(n - 1) + fibonacci(n - 2)

# Example Usage:
n = int(input("Enter a positive integer to find its Fibonacci number: "))
fib_number = fibonacci(n)
print(f"The {n}th Fibonacci number is: {fib_number}")

# Explain the above code:
# The function 'fibonacci' calculates the nth Fibonacci number using recursion.
# It has a base case that handles the simplest inputs (0 and 1) directly.
# For all other values of n, it calls itself twice to compute the two preceding Fibonacci numbers,
# summing them to get the current Fibonacci number. This continues until the base case is reached.
```

```
PS C:\Users\hp\OneDrive\Desktop\AI_Assisted_Coding> & "c:\Users\hp\AppData\Local\Programs\Python\Python314\python.exe" "c:\Users\hp\OneDrive\Desktop\AI_Assisted_Coding\Assignment-5.5.py"
Enter a positive integer to find its Fibonacci number: 20
The 20th Fibonacci number is: 6765
```

Justification:

This task helps us understand **how recursion works clearly and transparently**.

- The AI-generated code includes **base cases**, which prevent infinite recursion.
- Comments explain when the function stops and when it calls itself again.

By adding clear explanations, the code becomes easier to understand and debug.

This improves **trust, learning, and correctness**, which are important in ethical AI coding. Task 3:

Transparency in Error Handling

PROMPT:

Generate a Python program that reads a file and processes data. Generate code with proper error handling and clear explanations for each exception. Code with meaningful exception handling. Clear comments explaining each error scenario. Validation that explanations align with runtime behavior.

CODE and OUTPUT:

```
68 # Generate a Python program that reads a file and processes data. Generate code with proper error handling and clear explanations for each exception. Code with no
69 # Python Program to Read a File and Process Data with Error Handling
70 def read_and_process_file(file_path):
71     try:
72         # Attempt to open the file
73         with open(file_path, 'r') as file:
74             data = file.readlines()
75             # Process the data (for example, print each line)
76             for line in data:
77                 print(line.strip())
78     except FileNotFoundError:
79         # This exception is raised when the specified file does not exist
80         print(f"Error: The file '{file_path}' was not found. Please check the file path and try again.")
81     except PermissionError:
82         # This exception is raised when there are insufficient permissions to read the file
83         print(f"Error: You do not have permission to read the file '{file_path}'. Please check your permissions.")
84     except IsADirectoryError:
85         # This exception is raised when a directory is provided instead of a file
86         print(f"Error: The path '{file_path}' is a directory, not a file. Please provide a valid file path.")
87     except Exception as e:
88         # Catch all for any other exceptions that may occur
89         print(f"An unexpected error occurred: {e}")
90 # Example Usage
91 file_path = 'example.txt' # Replace with your file path
92 read_and_process_file(file_path)
```

```
60320' --' 'D:\AI\AI-5.5.py'
Error: The file 'example.txt' was not found. Please check the file path and try again.
C:\PS D:\AI>
```

Justification:

Proper error handling is important for safe and reliable software.

- The program handles common errors like missing files and permission issues.
- Each exception is explained so users know what went wrong.

This task shows that AI-generated code must be **checked and improved** to avoid crashes and confusion.

Clear error messages increase transparency and user confidence. Task 4:

Security in User Authentication

PROMPT:

Generate a Python-based login system. Check whether the AI uses secure password handling practices. Identification of security flaws (plain-text passwords, weak validation). Revised version using password hashing and input validation. Short note on best practices for secure authentication.

CODE and OUTPUT:

```

94 #Generate a Python-based login system.Check whether the AI uses secure password handling practices.Identification of security flaws (plain-text passwords, weak
95 import hashlib
96 import re
97 # Simple Login System with Secure Password Handling
98 class LoginSystem:
99     def __init__(self):
100         # In a real application, user data would be stored in a database
101         self.users = {}
102
103     def hash_password(self, password):
104         # Hash the password using SHA-256 for secure storage
105         return hashlib.sha256(password.encode()).hexdigest()
106
107     def validate_password(self, password):
108         # Validate password strength: at least 8 characters, including letters and numbers
109         if len(password) < 8:
110             return False
111         if not re.search("[a-zA-Z]", password):
112             return False
113         if not re.search("[0-9]", password):
114             return False
115         return True
116
117     def register(self, username, password):
118         if username in self.users:
119             print("Error: Username already exists.")
120             return
121         if not self.validate_password(password):
122             print("Error: Password must be at least 8 characters long and include both letters and numbers.")
123             return
124         hashed_password = self.hash_password(password)
125         self.users[username] = hashed_password
126         print("User registered successfully.")
127
128     def login(self, username, password):
129         if username not in self.users:
130             print("Error: Username does not exist.")
131             return
132         hashed_password = self.hash_password(password)
133         if self.users[username] == hashed_password:
134             print("Login successful.")
135         else:
136             print("Error: Incorrect password.")
137
138 # Example Usage
139 login_system = LoginSystem()
140 login_system.register("user1", "Password123")
141 login_system.login("user1", "Password123")

```

```

119         print("Error: Username already exists.")
120         return
121     if not self.validate_password(password):
122         print("Error: Password must be at least 8 characters long and include both letters and numbers.")
123         return
124     hashed_password = self.hash_password(password)
125     self.users[username] = hashed_password
126     print("User registered successfully.")
127
128     def login(self, username, password):
129         if username not in self.users:
130             print("Error: Username does not exist.")
131             return
132         hashed_password = self.hash_password(password)
133         if self.users[username] == hashed_password:
134             print("Login successful.")
135         else:
136             print("Error: Incorrect password.")
137
138 # Example Usage
139 login_system = LoginSystem()
140 login_system.register("user1", "Password123")
141 login_system.login("user1", "Password123")
142
143 # Best Practices for Secure Authentication:
144 # 1. Always hash passwords before storing them to prevent plain-text password storage.
145 # 2. Use strong password policies to ensure users create secure passwords.
146 # 3. Implement account lockout mechanisms to prevent brute-force attacks.
147 # 4. Use secure protocols (like HTTPS) to protect data in transit.
148 """
149
150 PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
151
152 PS D:\AI> & "c:\Users\atig\AppData\Local\Programs\Python\Python314\python.exe" "c:\Users\atig\.vscode\extensions\ms-python.debugpy-2025.18.8-win32-x64\
51163' '...' D:\AI\AI-5.5.py'
User registered successfully.
Login successful.
PS D:\AI>

```

Justification:

This task highlights security risks in AI-generated code.

- The insecure version stores passwords in plaintext, which is unsafe.

- The improved version uses **password hashing**, making it difficult for attackers to steal passwords.

Developers must review AI code carefully to ensure it follows **secure coding practices**. Security is a key ethical responsibility when handling user data.

Task 5: Privacy in Data Logging

PROMPT:

Generate a Python script that logs user activity (username, IP address, timestamp). Examine whether sensitive data is logged unnecessarily or insecurely. Identify privacy risks in logging. Improved version with minimal, anonymized, or masked logging. Explanation of privacy-aware logging principles.

CODE AND OUTPUT:

```

140 # Generate a Python script that logs user activity (username, IP address, timestamp). Examine whether sensitive data is logged unnecessarily or insecurely. Identify
141 # privacy risks in logging. Improved version with minimal, anonymized, or masked logging. Explanation of privacy-aware logging principles.
142
143 import logging
144
145 logging.basicConfig(
146     level=logging.INFO,
147     format='%(asctime)s - %(message)s'
148 )
149
150
151 def log_user_activity(username, ip_address):
152     # Anonymize IP address
153     anonymized_ip = ".".join(ip_address.split('.')[:-1]) + ['xxx']
154     logging.info(f'User: {username}, IP: {anonymized_ip}')
155
156
157 log_user_activity('user1', '192.168.1.100')
158
159 # Explanation of Privacy-aware Logging Principles:
160 # 1. Minimize Data Collection: Only log information that is necessary for the intended purpose
161 #    (e.g., username and anonymized IP address) to reduce privacy risks.
162 # 2. Anonymization: Mask or anonymize sensitive data (like IP addresses) to protect user privacy.
163 # 3. Secure Storage: Ensure that log files are stored securely with appropriate access controls
164 #    to prevent unauthorized access.
165 # 4. Compliance: Adhere to data protection regulations and best practices regarding user data logging.
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

```

```

2025-01-30 10:24:05,795 - User: user1, IP: 192.168.1.xxx
(Python 3.11.7)

```

Justification:

This task focuses on **user privacy and data protection**.

- Logging full usernames and IP addresses can expose sensitive information.
- The improved version masks user data and logs only what is necessary.

This follows the principle of **data minimization**, which helps protect user privacy.

Ethical AI coding requires respecting personal data and avoiding unnecessary data collection.