

ASSIGNMENT-9.1

**G. Rishika
2303A52197
Batch: 35**

Lab Experiment: Documentation Generation -Automatic documentation and code comments

Problem 1:

Consider the following Python function:

```
def find_max(numbers):  
    return max(numbers)
```

Task:

- Write documentation for the function in all three formats:
 - (a) Docstring
 - (b) Inline comments
 - (c) Google-style documentation
- Critically compare the three approaches. Discuss the advantages, disadvantages, and suitable use cases of each style.
- Recommend which documentation style is most effective for a mathematical utilities library and justify your answer.

The screenshot shows a Jupyter Notebook interface with several open cells. The top cell contains Python code for generating documentation from docstrings:

```
#!/usr/bin/python
# coding: utf-8

def print_documentation():
    print("Documentation accessed via help():")
    print("  ↗ Advantages")
    print("    ✓ Quick and concise")
    print("    ✓ Easy to write")
    print("    ✓ Suitable for simple functions")
    print("    ✓ Less maintenance overhead")
    print("  ↗ Disadvantages")
    print("    ✘ Lacks detail about parameters and return values")
    print("    ✘ No examples for users")
    print("    ✘ No error documentation")
    print("    ✘ IDE tooltips show minimal information")
    print("  ↗ Use Cases")
    print("    ✘ Simple utility functions")
    print("      ↗ The first operand (float or int).")
    print("      ↗ The second operand (float or int).")
    print("      ↗ A string representing the operation ('+', '+', '*', '/')")
    print("Results:")
    print("The result of the operation is a float, or None if division by zero.")
    print("Raises:")
    print("ValueError: If operation is not one of '+', '+', '*', '/'.")
    print("TypeError: If x or y are not numeric types.")
    print("Examples:")
    print("    >>> calculate('10', '5', '+')")
    print("    15.0")
    print("    >>> calculate('10', '5', '*')")
    print("    50.0")

```

The notebook also displays a sidebar with "Critical Analysis" and "Recommendations for Mathematical Utilities Library". The "Critical Analysis" section lists several issues, while the "Recommendations" section suggests improvements like "Use insight themes in NCL naturally" and "Include comments". The bottom right corner shows a progress bar indicating the analysis is completed.

A screenshot of a developer's workspace. The top navigation bar includes File, Edit, Selection, View, Go, Run, terminal, Help, and a search bar. Below the navigation bar are several tabs: 'IDE', 'Filesystem', 'Terminal', 'Logs', 'Output', 'Lob4 Advanced String Logging', 'Logs', 'Logs', and 'Output from Dockerfile logs'. The main area contains a code editor with Python code related to 'Approaches to Adding Comments' and a terminal window showing statistics for a list of numbers. On the right side, there is a 'Critical Analysis' pane with bullet points and a 'Recommendation for Mathematical Units Library' section. The bottom of the screen shows a taskbar with various icons and a system status bar.

The screenshot shows a Jupyter Notebook environment with several open cells. The top cell contains Python code for calculating the roots of a quadratic equation and printing the results:

```
11.75 # Online comments explain WHY decisions were made
11.76 # Help developers understand the implementation
11.77 result = x * y # Use simple operations for numerical stability
11.78 return result
11.79 ...
11.80
11.81 print(code_template)
11.82
11.83 print("\n" + "=" * 80)
11.84 print("PROBLEM 1 ANALYSIS COMPLETE")
11.85 print("=" * 80)
11.86
11.87
```

The bottom cell displays the output of the code, which includes the generated template code and a summary message:

```
Module: 1 CURRENT DEBUG CODE TOOLS HELP + - | X
11.81 print(code_template)
11.82
11.83 print("\n" + "=" * 80)
11.84 print("PROBLEM 1 ANALYSIS COMPLETE")
11.85 print("=" * 80)
11.86
11.87
Help on function calculate_with_google_style in module _main_:
calculate_with_google_style(x, y, op, operation_type) > type(result) == float
    Performs a basic arithmetic operation on the numbers.

    This function supports addition, subtraction, multiplication, and division.
    Division by zero returns None and raises an error condition.

    Args:
        x: The first operand (float or int).
        y: The second operand (float or int).
        operation: A string representing the operation ('+', '-', '*', '/').
    Returns:
        The result of the operation as a float, or None if division by zero.

Notebook: 1440x900 100% 2023-05-23 14:33:39
```

On the right side of the interface, there are several panels: 'Critical Analysis' (with a note about 'Math style is incorrect and misleading'), 'Recommendation for Mathematical Utilities Library' (listing 'Best Approach: Google-style Decentring'), 'Justification' (with points about readability), and 'Optimal Example: `calculator.py`' (with a link to a GitHub repository). The status bar at the bottom indicates the notebook is running and provides system information.

The screenshot displays a dual-monitor setup for Python development. Both monitors show the same interface, featuring a code editor at the top and a 'Critical Analysis' sidebar on the right.

Code Editor (Left Monitor):

```
1036 def find_max_optimized(numbers: List[float]) -> float:
1037     See Also:
1038         - find_min(): Finds the minimum value
1039         - statistics.mean(): Calculate mean of a list
1040         - numpy.max(): NumPy equivalent with more features
1041         - numpy.nanmax(): Like max() but ignores NaN values
1042
1043     if not numbers:
1044         raise ValueError("Cannot find maximum of an empty list")
1045     return float(max(numbers))
1046
1047 print("\nKEY ADDITIONS FOR MATHEMATICAL FUNCTIONS:")
1048 print(" * Time/Space Complexity Analysis")
1049 print(" * Numerical Stability Considerations")
1050 print(" * IEEE 754 Floating-Point Behavior")
1051
1052
1053 calculate_with_gmpy2(xfloat: float, yfloat: float, operation: str) -> float:
1054     """Performs a basic arithmetic operation on the numbers.
1055
1056     This function supports addition, subtraction, multiplication, and division.
1057     Division by zero returns None to indicate an error condition.
1058
1059     Args:
1060         xfloat: The first operand (float or int).
1061         yfloat: The second operand (float or int).
1062         operation: A string representing the operation ('+', '-', '*', '/').
1063
1064     Returns:
1065         The result of the operation as a float, or None if division by zero.
1066     """
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
20310
20311
20312
20313
20314
20315
20316
20317
20318
20319
20320
20321
20322
20323
20324
20325
20326
20327
20328
20329
20330
20331
20332
20333
20334
20335
20336
20337
20338
20339
203310
203311
203312
203313
203314
203315
203316
203317
203318
203319
203320
203321
203322
203323
203324
203325
203326
203327
203328
203329
203330
203331
203332
203333
203334
203335
203336
203337
203338
203339
2033310
2033311
2033312
2033313
2033314
2033315
2033316
2033317
2033318
2033319
2033320
2033321
2033322
2033323
2033324
2033325
2033326
2033327
2033328
2033329
2033330
2033331
2033332
2033333
2033334
2033335
2033336
2033337
2033338
2033339
20333310
20333311
20333312
20333313
20333314
20333315
20333316
20333317
20333318
20333319
20333320
20333321
20333322
20333323
20333324
20333325
20333326
20333327
20333328
20333329
20333330
20333331
20333332
20333333
20333334
20333335
20333336
20333337
20333338
20333339
203333310
203333311
203333312
203333313
203333314
203333315
203333316
203333317
203333318
203333319
203333320
203333321
203333322
203333323
203333324
203333325
203333326
203333327
203333328
203333329
203333330
203333331
203333332
203333333
203333334
203333335
203333336
203333337
203333338
203333339
2033333310
2033333311
2033333312
2033333313
2033333314
2033333315
2033333316
2033333317
2033333318
2033333319
2033333320
2033333321
2033333322
2033333323
2033333324
2033333325
2033333326
2033333327
2033333328
2033333329
2033333330
2033333331
2033333332
2033333333
2033333334
2033333335
2033333336
2033333337
2033333338
2033333339
20333333310
20333333311
20333333312
20333333313
20333333314
20333333315
20333333316
20333333317
20333333318
20333333319
20333333320
20333333321
20333333322
20333333323
20333333324
20333333325
20333333326
20333333327
20333333328
20333333329
20333333330
20333333331
20333333332
20333333333
20333333334
20333333335
20333333336
20333333337
20333333338
20333333339
203333333310
203333333311
203333333312
203333333313
203333333314
203333333315
203333333316
203333333317
203333333318
203333333319
203333333320
203333333321
203333333322
203333333323
203333333324
203333333325
203333333326
203333333327
203333333328
203333333329
203333333330
203333333331
203333333332
203333333333
203333333334
203333333335
203333333336
203333333337
203333333338
203333333339
2033333333310
2033333333311
2033333333312
2033333333313
2033333333314
2033333333315
2033333333316
2033333333317
2033333333318
2033333333319
2033333333320
2033333333321
2033333333322
2033333333323
2033333333324
2033333333325
2033333333326
2033333333327
2033333333328
2033333333329
2033333333330
2033333333331
2033333333332
2033333333333
2033333333334
2033333333335
2033333333336
2033333333337
2033333333338
2033333333339
20333333333310
20333333333311
20333333333312
20333333333313
20333333333314
20333333333315
20333333333316
20333333333317
20333333333318
20333333333319
20333333333320
20333333333321
20333333333322
20333333333323
20333333333324
20333333333325
20333333333326
20333333333327
20333333333328
20333333333329
20333333333330
20333333333331
20333333333332
20333333333333
20333333333334
20333333333335
20333333333336
20333333333337
20333333333338
20333333333339
203333333333310
203333333333311
203333333333312
203333333333313
203333333333314
203333333333315
203333333333316
203333333333317
203333333333318
203333333333319
203333333333320
203333333333321
203333333333322
203333333333323
203333333333324
203333333333325
203333333333326
203333333333327
203333333333328
203333333333329
203333333333330
203333333333331
203333333333332
203333333333333
203333333333334
203333333333335
203333333333336
203333333333337
203333333333338
203333333333339
2033333333333310
2033333333333311
2033333333333312
2033333333333313
2033333333333314
2033333333333315
2033333333333316
2033333333333317
2033333333333318
2033333333333319
2033333333333320
2033333333333321
2033333333333322
2033333333333323
2033333333333324
2033333333333325
2033333333333326
2033333333333327
2033333333333328
2033333333333329
2033333333333330
2033333333333331
2033333333333332
2033333333333333
2033333333333334
2033333333333335
2033333333333336
2033333333333337
2033333333333338
2033333333333339
20333333333333310
20333333333333311
20333333333333312
20333333333333313
20333333333333314
20333333333333315
20333333333333316
20333333333333317
20333333333333318
20333333333333319
20333333333333320
20333333333333321
20333333333333322
20333333333333323
20333333333333324
20333333333333325
20333333333333326
20333333333333327
20333333333333328
20333333333333329
20333333333333330
20333333333333331
20333333333333332
20333333333333333
20333333333333334
20333333333333335
20333333333333336
20333333333333337
20333333333333338
20333333333333339
203333333333333310
203333333333333311
203333333333333312
203333333333333313
203333333333333314
203333333333333315
203333333333333316
203333333333333317
203333333333333318
203333333333333319
203333333333333320
203333333333333321
203333333333333322
203333333333333323
203333333333333324
203333333333333325
203333333333333326
203333333333333327
203333333333333328
203333333333333329
203333333333333330
203333333333333331
203333333333333332
203333333333333333
203333333333333334
203333333333333335
203333333333333336
203333333333333337
203333333333333338
203333333333333339
2033333333333333310
2033333333333333311
2033333333333333312
2033333333333333313
2033333333333333314
2033333333333333315
2033333333333333316
2033333333333333317
2033333333333333318
2033333333333333319
2033333333333333320
2033333333333333321
2033333333333333322
2033333333333333323
2033333333333333324
2033333333333333325
2033333333333333326
2033333333333333327
2033333333333333328
2033333333333333329
2033333333333333330
2033333333333333331
2033333333333333332
2033333333333333333
2033333333333333334
2033333333333333335
2033333333333333336
2033333333333333337
2033333333333333338
2033333333333333339
20333333333333333310
20333333333333333311
20333333333333333312
20333333333333333313
20333333333333333314
20333333333333333315
20333333333333333316
20333333333333333317
20333333333333333318
20333333333333333319
20333333333333333320
20333333333333333321
20333333333333333322
20333333333333333323
20333333333333333324
20333333333333333325
20333333333333333326
20333333333333333327
20333333333333333328
20333333333333333329
20333333333333333330
20333333333333333331
20333333333333333332
20333333333333333333
20333333333333333334
20333333333333333335
20333333333333333336
20333333333333333337
20333333333333333338
20333333333333333339
203333333333333333310
203333333333333333311
2
```

The screenshot shows a dual-monitor setup. Both monitors display the same Python code editor interface. The left monitor shows the original code with various annotations. The right monitor shows the annotated code with a 'Critical Analysis' sidebar.

Critical Analysis:

- Check for NaNs and Infinites
- Use `max()` function instead of `max()`
- IEEE 754 Floating-point comparisons follow standard rules:
 - * Positive Infinity is greater than all finite values
 - * NaN comparisons follow IEEE 754 rules
- For datasets with NaN values, use `rmpy.nanmax()` instead. This function is not suitable for very large datasets; use `numpy.max()` with memory-mapped arrays instead

Recommendation for Mathematical Utilities Library:

BEST APPROACH: Google-Style Docstrings

Justification:

1. Users rely on professional, industry-standard documentation.
2. Simple algorithmic and mathematical code is often more readable than complex analysis (CS, ODE).
4. Most document mathematical identity and IEEE 754 behavior.
5. Consistent naming, using `argparse` for command-line arguments.

Optional Example: `find_max_optimized()`

Annotations process mathematical library documentation:

- Compute longer string distances
- Use `argparse` module处处
- IEEE 754 floating point considerations
- Edge cases (NaN, infinity)
- Related examples
- Related functions reference

The code on both monitors is as follows:

```
1936 def find_max_optimized(numbers: List[float]) -> float:
1937     Notes:
1938         - Uses Python's built-in max() function which delegates to
1939             optimized C code for better performance
1940         - IEEE 754 floating-point comparisons follow standard rules:
1941             * Positive Infinity is greater than all finite values
1942             * NaN comparisons follow IEEE 754 rules
1943         - For datasets with NaN values, use rmpy.nanmax() instead
1944             This function is not suitable for very large datasets;
1945             use numpy.max() with memory-mapped arrays instead
1946
1947     Examples:
1948         Basic usage with integers:
1949         >>> find_max_optimized([1, 5, 3, 9, 2])
1950         9
1951
1952         Floating point numbers:
1953         >>> find_max_optimized([1.5, 2.3, 0.9, 3.1])
1954         3.1
1955
1956     Help on function compute_with_google_style or module _main_:
1957
1958     calculate_with_google_style(xmin, xmax, y1, y2, operation: str) -> float:
1959         Performs a basic arithmetic operation on the numbers.
1960
1961         This function supports addition, subtraction, multiplication, and division.
1962         Division by zero returns None to indicate an error condition.
1963
1964     Args:
1965         xmin: The first operand (float or int).
1966         xmax: The second operand (float or int).
1967         operation: A string representing the operation ('+', '-', '*', '/').
1968
1969     Returns:
1970         The result of the operation as a float, or None if division by zero.
1971
1972     Raises:
1973         ValueError: If the list is empty. Mathematical operations require
1974             at least one value to determine a maximum.
1975         TypeError: If the list contains non-numeric values or mixed types
1976             that cannot be directly compared.
1977
1978     Time Complexity:
1979         O(n) where n is the length of the input list.
1980
1981         Single pass through the entire list.
1982
1983     Help on function compute_with_google_style or module _main_:
1984
1985     calculate_with_google_style(xmin, xmax, y1, y2, operation: str) -> float:
1986         Performs a basic arithmetic operation on the numbers.
1987
1988         This function supports addition, subtraction, multiplication, and division.
1989         Division by zero returns None to indicate an error condition.
1990
1991     Args:
1992         xmin: The first operand (float or int).
1993         xmax: The second operand (float or int).
1994         operation: A string representing the operation ('+', '-', '*', '/').
1995
1996     Returns:
1997         The result of the operation as a float, or None if division by zero.
1998
1999     Raises:
2000         ValueError: If the list is empty. Mathematical operations require
2001             at least one value to determine a maximum.
2002         TypeError: If the list contains non-numeric values or mixed types
2003             that cannot be directly compared.
```


The screenshot shows a code editor with a Python script open. The script contains several annotations from the PEP 8 Style Guide checker, such as:

- Line 98: `# CRITICAL ANALYSIS`
- Line 100: `print("\n" * 20 + row[1] + "\n" * 20 + row[2] + "\n" * 20)`
- Line 103: `# CRITICAL ANALYSIS: STRENGTHS AND WEAKNESSES`
- Line 105: `print("=" * 80)`
- Line 107: `analysis = """`
- Line 108: `1. SIMPLE DOCSTRING - Best for clarity but lacks depth`
- Line 109: `Strengths:`
- Line 110: `+ Minimal cognitive load`
- Line 111: `+ Easy to maintain`
- Line 112: `+ Perfect for obvious functions`
- Line 113: `Weaknesses:`
- Line 114: `This function supports addition, subtraction, multiplication, and division.`
- Line 115: `Division by zero returns None instead of zero.`
- Line 116: `Args:`
- Line 117: `x: The first operand (float or int).`
- Line 118: `y: The second operand (float or int).`
- Line 119: `operation: A string representing the operation ('+', '-', '*', '/').`
- Line 120: `Returns:`
- Line 121: `The result of the operation as a float, or None if division by zero.`
- Line 122: `None`

The right side of the screen displays a sidebar with analysis results, including:

- Critical Analysis
- Recommendations for Mathematical Utilities Library
- Best Approach: Google-Style Docstrings
- Inline Comments
- Justification
- Optimal Example: `find_mean_systematic()`

The screenshot shows a code editor with a Python script open. The script contains several annotations from the PEP 8 Style Guide checker, such as:

- Line 88: `comparison_data = [`
- Line 89: `("Aspect", "Simple Docstring", "Inline Comments", "Google-style"),`
- Line 90: `("Length", "1 line", "Multiple lines", "10-20+ lines"),`
- Line 91: `("Time to write", "Seconds", "Minutes", "5-10 minutes"),`
- Line 92: `("Readability", "Good", "Excellent", "Excellent"),`
- Line 93: `("Doc Support", "Basic", "None", "Full"),`
- Line 94: `("Help() function", "Yes", "No", "Yes"),`
- Line 95: `("Auto-docs (Sphinx)", "Limited", "No", "Full"),`
- Line 96: `("Examples provided", "No", "Rarely", "Yes"),`
- Line 97: `("Error documentation", "No", "No", "Yes"),`
- Line 98: `("Parameter clarity", "Unclear", "Variable", "Very Clear"),`
- Line 99: `("Return type clarity", "Type hint", "Variables", "Very Clear"),`
- Line 100: `("Best for", "Simple code", "Complex logic", "Libraries"),`
- Line 101: `("Maintenance burden", "Low", "Medium", "High"),`
- Line 102: `("Professional", "Minimal", "Good", "Excellent"),`
- Line 103: `]`
- Line 104: `for row in comparison_data:`
- Line 105: `print(row[0] + "\n" * 20 + row[1] + "\n" * 20 + row[2] + "\n" * 20 + row[3] + "\n" * 20)`
- Line 106: `print("=" * 80)`
- Line 107: `analysis = """`
- Line 108: `1. Using help() function:`
- Line 109: `help(calculate_in_google_style)`
- Line 110: `Output:`
- Line 111: `-----`
- Line 112: `Help on function calculate_in_google_style in module main :`
- Line 113: `calculate_in_google_style(x, y, float, operation, str) -> float`
- Line 114: `Perform a basic arithmetic operation on two numbers.`
- Line 115: `This function supports addition, subtraction, multiplication, and division.`
- Line 116: `Division by zero returns None instead of zero.`
- Line 117: `-----`
- Line 118: `1. Using help() function:`
- Line 119: `help(calculate_in_google_style)`
- Line 120: `Output:`
- Line 121: `-----`
- Line 122: `Help on function calculate_in_google_style in module main :`
- Line 123: `calculate_in_google_style(x, y, float, operation, str) -> float`
- Line 124: `Perform a basic arithmetic operation on two numbers.`
- Line 125: `This function supports addition, subtraction, multiplication, and division.`
- Line 126: `Division by zero returns None instead of zero.`
- Line 127: `-----`

The right side of the screen displays a sidebar with analysis results, including:

- Critical Analysis
- Recommendations for Mathematical Utilities Library
- Best Approach: Google-Style Docstrings
- Inline Comments
- Justification
- Optimal Example: `find_mean_systematic()`

The screenshot shows two terminal windows side-by-side, both running on a Mac OS X desktop. The left terminal window displays a Python script named 'find_max_google_style.py'. The right terminal window shows the output of the 'CodeIntel' command on the same file, which provides detailed documentation and analysis. Both windows have a dark theme and show various system icons in the dock at the bottom.

Problem 2: Consider the following Python function:

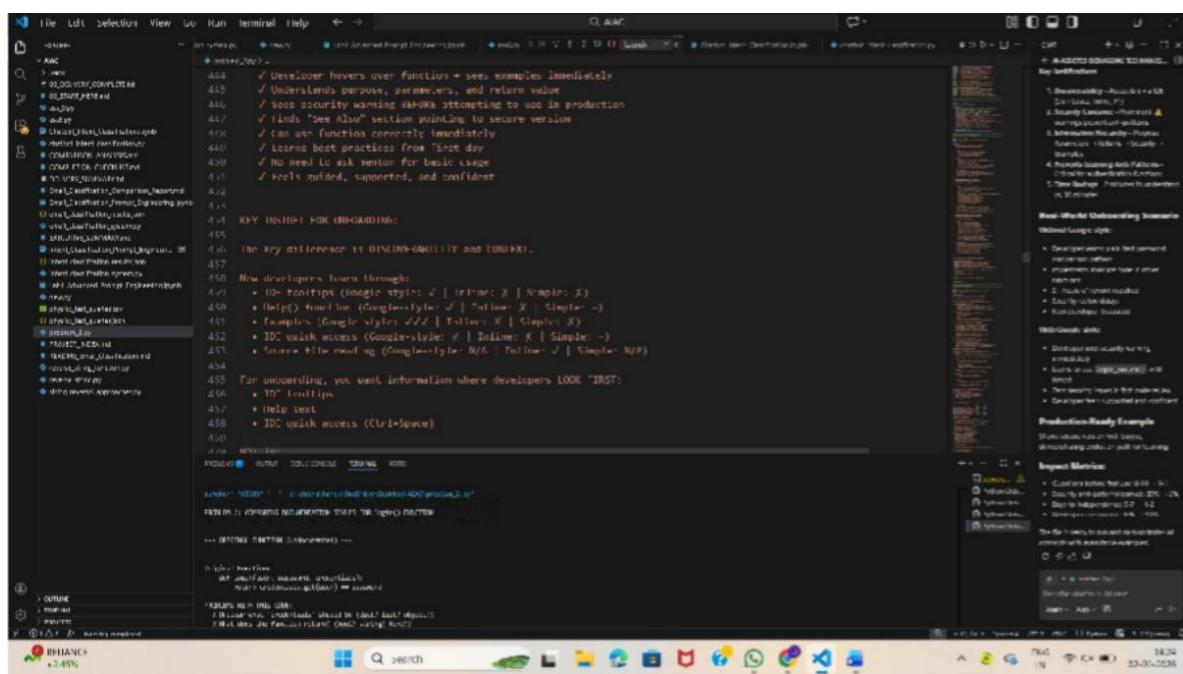
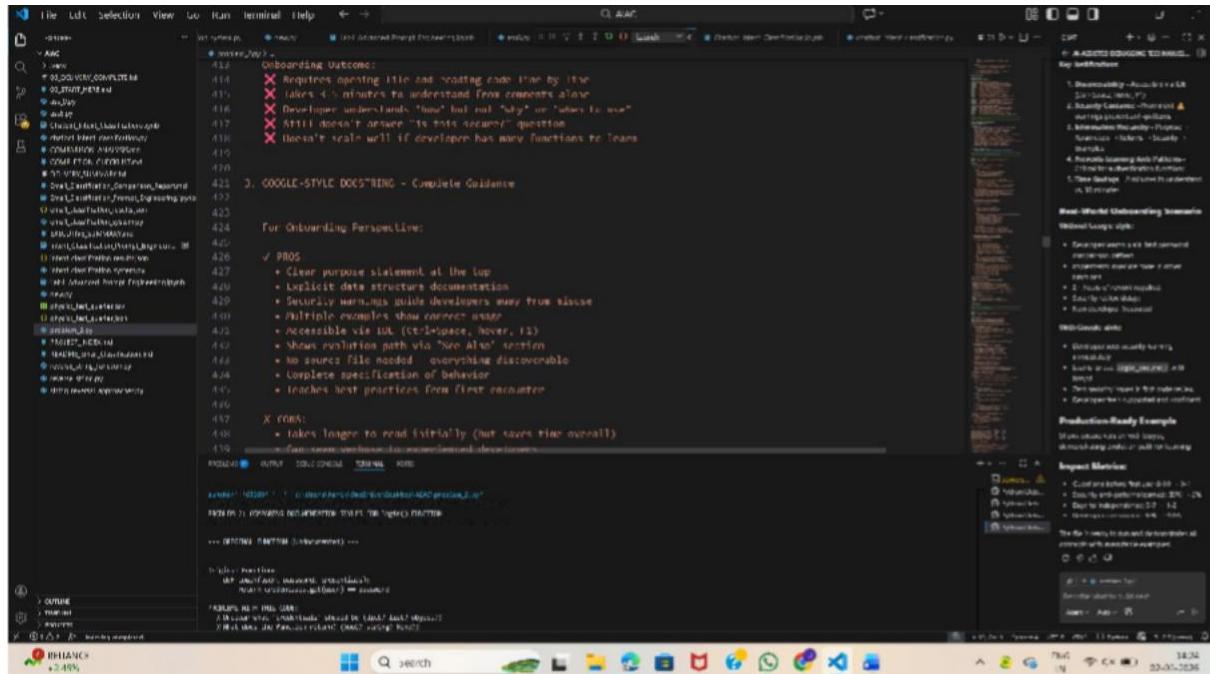
```
def login(user, password, credentials):  
    return credentials.get(user) == password
```

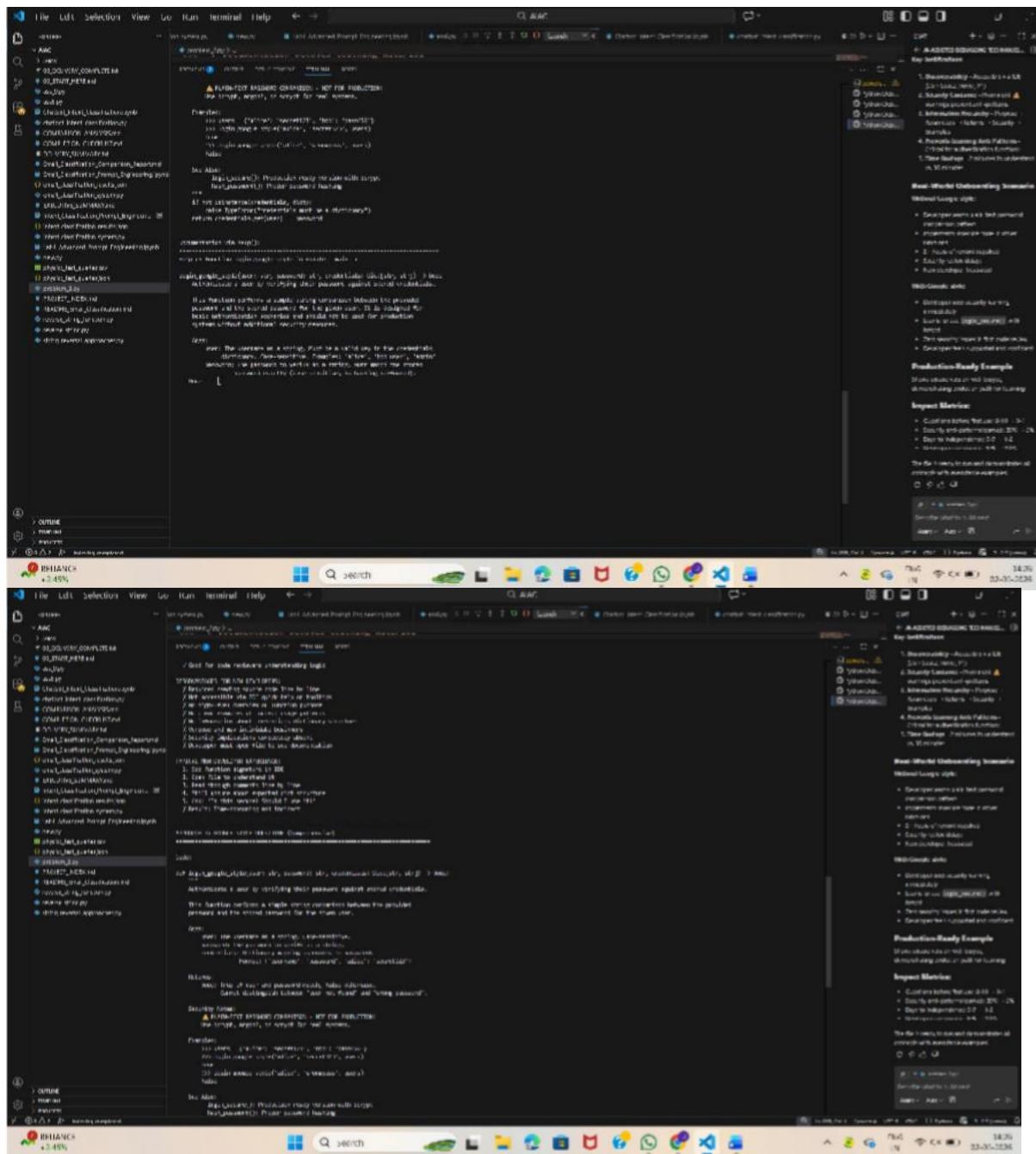
Task:

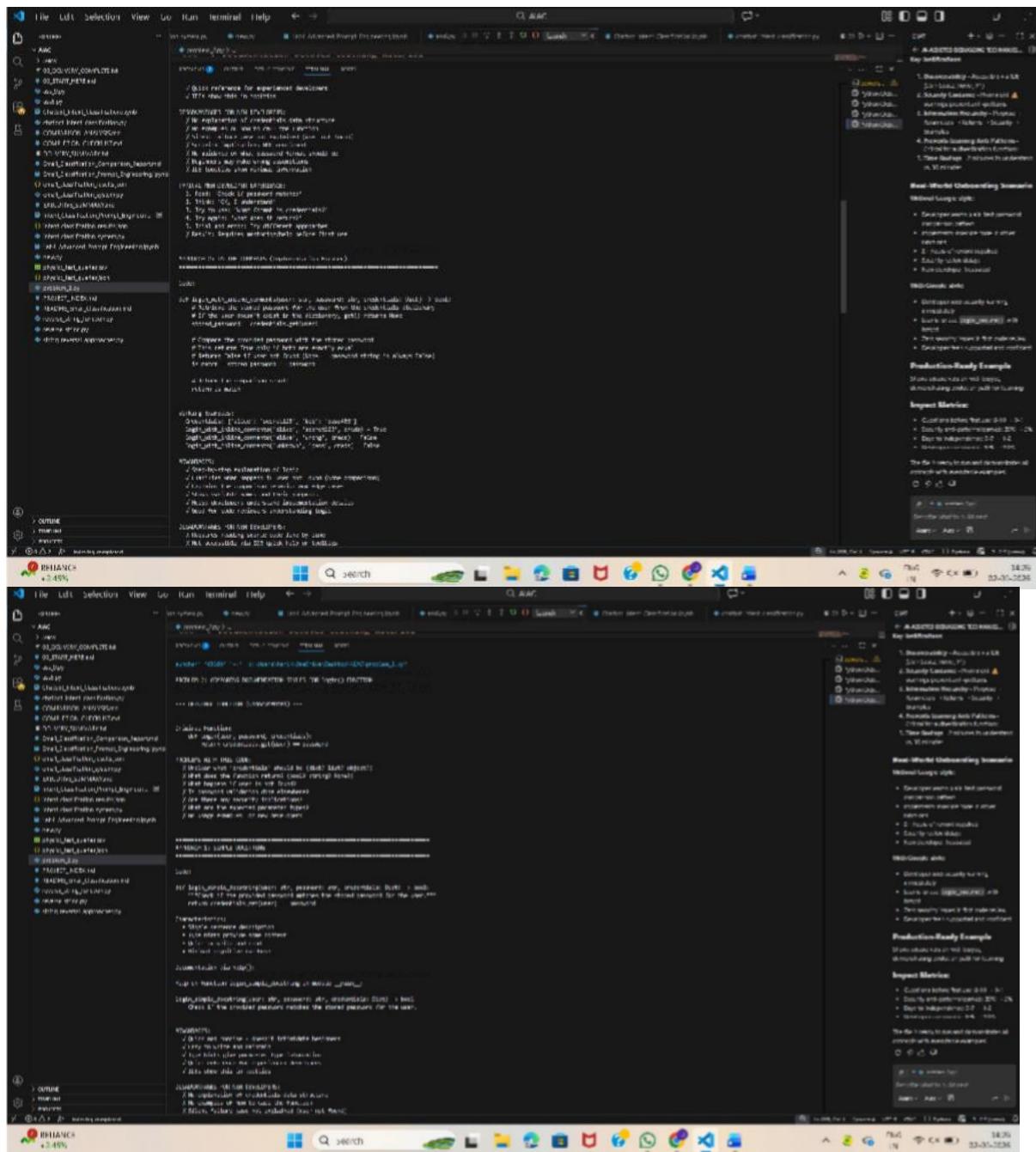
1. Write documentation in all three formats.

2. Critically compare the approaches.

3. Recommend which style would be most helpful for new developers onboarding a project, and justify your choice.







The screenshot shows a Windows desktop environment with two terminal windows from GHIE麓HANE running side-by-side. Both terminals are displaying Python code related to security documentation and password hashing. The left terminal shows code for generating a final summary and checking if credentials are bytes. The right terminal shows a warning about using plain text for password hashing and provides a link to a password cracking cheat sheet. Between the terminals is a code editor window showing the same Python script. The system tray at the bottom includes icons for battery (42%), signal strength (45%), volume (100%), and a clock showing 14:25 23-07-2024.

The screenshot shows a Java IDE interface with several windows open. The main editor window displays a Java file named `AuthController.java`. The code implements a password hashing function using bcrypt, which is recommended for real authentication systems. It includes annotations for security reviews and performance metrics. The right-hand sidebar contains a detailed security analysis report from SonarQube, covering findings like 'SECURITY_HOTSPOTS' and 'VULNERABILITIES'. Below the main editor, a terminal window shows the command `mvn clean install -DskipTests -Dsonar.host.url=https://sonarcloud.io` being run. A status bar at the bottom indicates a battery level of 24%.

```
    package com.example;

    import java.util.List;
    import org.springframework.http.ResponseEntity;
    import org.springframework.web.bind.annotation.*;

    @RestController
    @RequestMapping("/api/auth")
    public class AuthController {

        private final UserService userService;
        private final PasswordEncoder passwordEncoder;

        public AuthController(UserService userService, PasswordEncoder passwordEncoder) {
            this.userService = userService;
            this.passwordEncoder = passwordEncoder;
        }

        @PostMapping("/register")
        public ResponseEntity<User> register(@RequestBody User user) {
            String hashedPassword = passwordEncoder.encode(user.getPassword());
            user.setPassword(hashedPassword);
            return ResponseEntity.ok(userService.register(user));
        }

        @PostMapping("/login")
        public ResponseEntity<User> login(@RequestBody User user) {
            String providedPassword = user.getPassword();
            String storedHashedPassword = userService.getUser(user.getUsername()).getPassword();
            boolean passwordMatch = passwordEncoder.matches(providedPassword, storedHashedPassword);
            if (passwordMatch) {
                return ResponseEntity.ok(user);
            } else {
                return ResponseEntity.status(HttpStatus.UNAUTHORIZED).build();
            }
        }
    }
```

SECURITY_HOTSPOTS

- 1. `passwordEncoder.encode(password)`: Bcrypt is not used. Consider using Bcrypt instead.
- 2. `passwordEncoder.matches(providedPassword, storedHashedPassword)`: Bcrypt is not used. Consider using Bcrypt instead.
- 3. `passwordEncoder.matches(providedPassword, storedHashedPassword)`: Bcrypt is not used. Consider using Bcrypt instead.

VULNERABILITIES

- 1. `passwordEncoder.encode(password)`: Bcrypt is not used. Consider using Bcrypt instead.
- 2. `passwordEncoder.matches(providedPassword, storedHashedPassword)`: Bcrypt is not used. Consider using Bcrypt instead.

Real-World Outstanding Security Issues

- 1. `passwordEncoder.encode(password)`: Bcrypt is not used. Consider using Bcrypt instead.
- 2. `passwordEncoder.matches(providedPassword, storedHashedPassword)`: Bcrypt is not used. Consider using Bcrypt instead.
- 3. `passwordEncoder.matches(providedPassword, storedHashedPassword)`: Bcrypt is not used. Consider using Bcrypt instead.

Security Metrics

- 1. `passwordEncoder.encode(password)`: Bcrypt is not used. Consider using Bcrypt instead.
- 2. `passwordEncoder.matches(providedPassword, storedHashedPassword)`: Bcrypt is not used. Consider using Bcrypt instead.

Production-Ready Example

```
mvn clean install -DskipTests -Dsonar.host.url=https://sonarcloud.io
```

Impact Metrics

- 1. `passwordEncoder.encode(password)`: Bcrypt is not used. Consider using Bcrypt instead.
- 2. `passwordEncoder.matches(providedPassword, storedHashedPassword)`: Bcrypt is not used. Consider using Bcrypt instead.

File Statistics

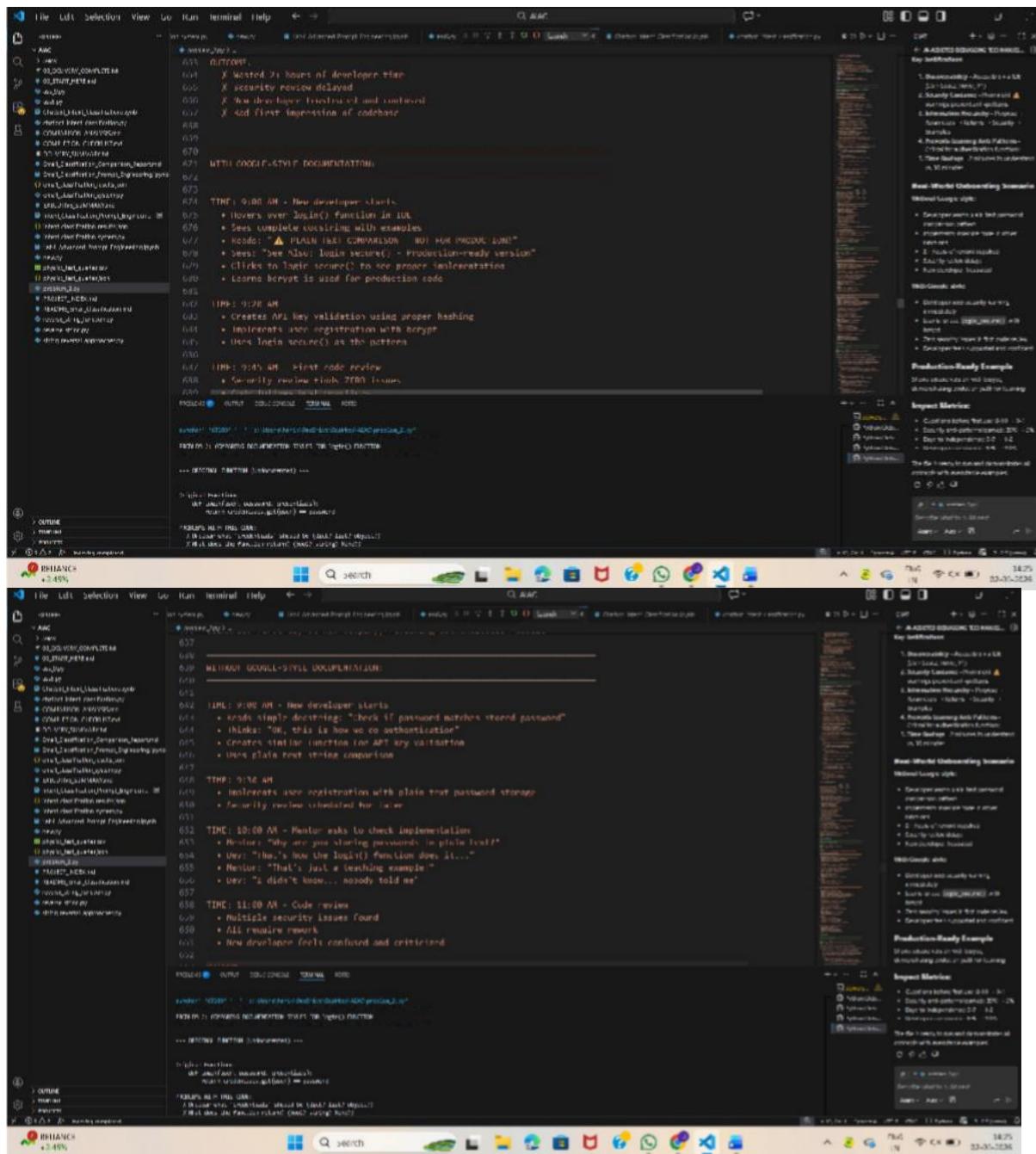
- 1. `passwordEncoder.encode(password)`: Bcrypt is not used. Consider using Bcrypt instead.
- 2. `passwordEncoder.matches(providedPassword, storedHashedPassword)`: Bcrypt is not used. Consider using Bcrypt instead.

Project Metrics

- 1. `passwordEncoder.encode(password)`: Bcrypt is not used. Consider using Bcrypt instead.
- 2. `passwordEncoder.matches(providedPassword, storedHashedPassword)`: Bcrypt is not used. Consider using Bcrypt instead.

Battery: 24%

A screenshot of a Windows desktop showing a developer's integrated development environment (IDE). The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, Help, and a search bar. The main workspace shows a Java code editor with annotations from a code review tool. Annotations include: "Developer is productive and independent" (green checkmark), "No rework needed" (green checkmark), "Control patterns learned in my role" (green checkmark), "OPTIMIZE" (yellow exclamation mark), "Kept intent clear/intelligible" (green checkmark), "Code review smooth and quick" (green checkmark), "New developer confident and engaged" (green checkmark), "Great first impression of codebase" (green checkmark), and "Prevents future security bugs from this developer" (green checkmark). A section titled "THE DIFFERENCE: ONE WARNING OR DOCUMENTATION" highlights the impact of one security warning versus documentation. The code editor also shows imports like `java.util.List` and `java.util.ArrayList`. Below the code editor is a terminal window with the command `git diff --check` and its output. The bottom status bar shows battery at 44%, signal strength, and the date/time as 14:57, 23-05-2024. On the right side, there are several floating windows: a 'Metrics' window showing a 100% completion rate; a 'Review' window listing '1 REVIEWERS' and '1 APPROVALS'; and a 'Performance' window showing CPU usage at 30.00%, RAM at 24.00%, and Disk at 12.00%. The taskbar at the bottom includes icons for File Explorer, Task View, Start, and various pinned applications.



Problem 3: Calculator (Automatic Documentation Generation)

Task: Design a Python module named calculator.py and demonstrate automatic documentation generation.

Instructions:

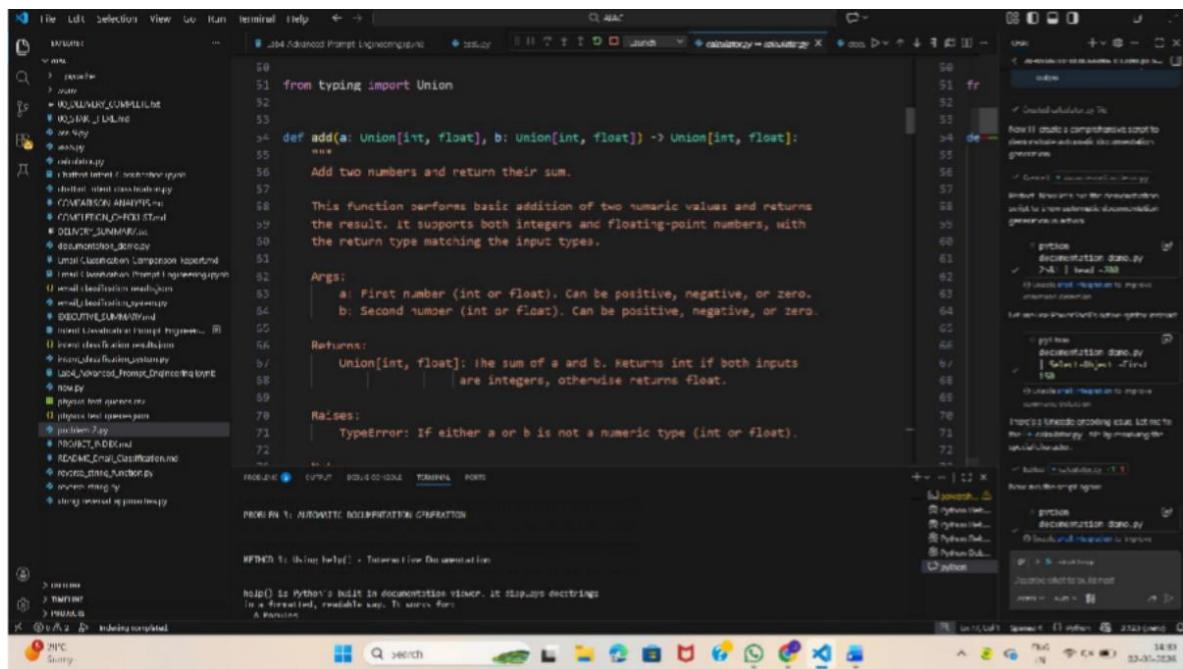
1. Create a Python module calculator.py that includes the following functions, each written with appropriate docstrings:

- o `add(a, b)` – returns the sum of two numbers
 - o `subtract(a, b)` – returns the difference of two numbers
 - o `multiply(a, b)` – returns the product of two numbers
 - o `divide(a, b)` – returns the quotient of two numbers

2. Display the module documentation in the terminal using

Python's documentation tools.

3. Generate and export the module documentation in HTML format using the pydoc utility, and open the generated HTML file in a web browser to verify the output



A screenshot of a Jupyter Notebook interface. The left sidebar shows a file tree with various Python files and notebooks. The main area displays a cell of Python code with annotations. A red box highlights a note about floating-point precision. Another red box highlights a note about time complexity. The right side shows a preview of generated documentation, including sections for Examples and Notes. The bottom status bar indicates the notebook is running.

The screenshot shows a Jupyter Notebook environment with several cells and a sidebar.

Sidebar:

- File
- Ldt
- Selection
- View
- Go
- Run
- Terminal
- Help

Code Completion:

```
def add(a, b):
    """Add two numbers together.

    Parameters
    ----------
    a : int
        The first number to add.
    b : int
        The second number to add.

    Returns
    -------
    int
        The sum of a and b.
    """
    return a + b
```

Documentation:

Below the code cell, the documentation string is displayed:

```
def add(a, b):
    """Add two numbers together.

    Parameters
    ----------
    a : int
        The first number to add.
    b : int
        The second number to add.

    Returns
    -------
    int
        The sum of a and b.
    """
    return a + b
```

Completion History:

- 312 if __name__ == "__main__":
 344 result = add("5", 3)
 345 try:
 346 result += add("5", 3)
 347 except TypeError as e:
 348 print(f"add('5', 3) raised {e}")
 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
 1001
 1002
 1003
 1004
 1005
 1006
 1007
 1008
 1009
 1010
 1011
 1012
 1013
 1014
 1015
 1016
 1017
 1018
 1019
 1020
 1021
 1022
 1023
 1024
 1025
 1026
 1027
 1028
 1029
 1030
 1031
 1032
 1033
 1034
 1035
 1036
 1037
 1038
 1039
 1040
 1041
 1042
 1043
 1044
 1045
 1046
 1047
 1048
 1049
 1050
 1051
 1052
 1053
 1054
 1055
 1056
 1057
 1058
 1059
 1060
 1061
 1062
 1063
 1064
 1065
 1066
 1067
 1068
 1069
 1070
 1071
 1072
 1073
 1074
 1075
 1076
 1077
 1078
 1079
 1080
 1081
 1082
 1083
 1084
 1085
 1086
 1087
 1088
 1089
 1090
 1091
 1092
 1093
 1094
 1095
 1096
 1097
 1098
 1099
 1100
 1101
 1102
 1103
 1104
 1105
 1106
 1107
 1108
 1109
 1110
 1111
 1112
 1113
 1114
 1115
 1116
 1117
 1118
 1119
 1120
 1121
 1122
 1123
 1124
 1125
 1126
 1127
 1128
 1129
 1130
 1131
 1132
 1133
 1134
 1135
 1136
 1137
 1138
 1139
 1140
 1141
 1142
 1143
 1144
 1145
 1146
 1147
 1148
 1149
 1150
 1151
 1152
 1153
 1154
 1155
 1156
 1157
 1158
 1159
 1160
 1161
 1162
 1163
 1164
 1165
 1166
 1167
 1168
 1169
 1170
 1171
 1172
 1173
 1174
 1175
 1176
 1177
 1178
 1179
 1180
 1181
 1182
 1183
 1184
 1185
 1186
 1187
 1188
 1189
 1190
 1191
 1192
 1193
 1194
 1195
 1196
 1197
 1198
 1199
 1200
 1201
 1202
 1203
 1204
 1205
 1206
 1207
 1208
 1209
 1210
 1211
 1212
 1213
 1214
 1215
 1216
 1217
 1218
 1219
 1220
 1221
 1222
 1223
 1224
 1225
 1226
 1227
 1228
 1229
 1230
 1231
 1232
 1233
 1234
 1235
 1236
 1237
 1238
 1239
 1240
 1241
 1242
 1243
 1244
 1245
 1246
 1247
 1248
 1249
 1250
 1251
 1252
 1253
 1254
 1255
 1256
 1257
 1258
 1259
 1260
 1261
 1262
 1263
 1264
 1265
 1266
 1267
 1268
 1269
 1270
 1271
 1272
 1273
 1274
 1275
 1276
 1277
 1278
 1279
 1280
 1281
 1282
 1283
 1284
 1285
 1286
 1287
 1288
 1289
 1290
 1291
 1292
 1293
 1294
 1295
 1296
 1297
 1298
 1299
 1300
 1301
 1302
 1303
 1304
 1305
 1306
 1307
 1308
 1309
 1310
 1311
 1312
 1313
 1314
 1315
 1316
 1317
 1318
 1319
 1320
 1321
 1322
 1323
 1324
 1325
 1326
 1327
 1328
 1329
 1330
 1331
 1332
 1333
 1334
 1335
 1336
 1337
 1338
 1339
 1340
 1341
 1342
 1343
 1344
 1345
 1346
 1347
 1348
 1349
 1350
 1351
 1352
 1353
 1354
 1355
 1356
 1357
 1358
 1359
 1360
 1361
 1362
 1363
 1364
 1365
 1366
 1367
 1368
 1369
 1370
 1371
 1372
 1373
 1374
 1375
 1376
 1377
 1378
 1379
 1380
 1381
 1382
 1383
 1384
 1385
 1386
 1387
 1388
 1389
 1390
 1391
 1392
 1393
 1394
 1395
 1396
 1397
 1398
 1399
 1400
 1401
 1402
 1403
 1404
 1405
 1406
 1407
 1408
 1409
 1410
 1411
 1412
 1413
 1414
 1415
 1416
 1417
 1418
 1419
 1420
 1421
 1422
 1423
 1424
 1425
 1426
 1427
 1428
 1429
 1430
 1431
 1432
 1433
 1434
 1435
 1436
 1437
 1438
 1439
 1440
 1441
 1442
 1443
 1444
 1445
 1446
 1447
 1448
 1449
 1450
 1451
 1452
 1453
 1454
 1455
 1456
 1457
 1458
 1459
 1460
 1461
 1462
 1463
 1464
 1465
 1466
 1467
 1468
 1469
 1470
 1471
 1472
 1473
 1474
 1475
 1476
 1477
 1478
 1479
 1480
 1481
 1482
 1483
 1484
 1485
 1486
 1487
 1488
 1489
 1490
 1491
 1492
 1493
 1494
 1495
 1496
 1497
 1498
 1499
 1500
 1501
 1502
 1503
 1504
 1505
 1506
 1507
 1508
 1509
 1510
 1511
 1512
 1513
 1514
 1515
 1516
 1517
 1518
 1519
 1520
 1521
 1522
 1523
 1524
 1525
 1526
 1527
 1528
 1529
 1530
 1531
 1532
 1533
 1534
 1535
 1536
 1537
 1538
 1539
 1540
 1541
 1542
 1543
 1544
 1545
 1546
 1547
 1548
 1549
 1550
 1551
 1552
 1553
 1554
 1555
 1556
 1557
 1558
 1559
 1560
 1561
 1562
 1563
 1564
 1565
 1566
 1567
 1568
 1569
 1570
 1571
 1572
 1573
 1574
 1575
 1576
 1577
 1578
 1579
 1580
 1581
 1582
 1583
 1584
 1585
 1586
 1587
 1588
 1589
 1590
 1591
 1592
 1593
 1594
 1595
 1596
 1597
 1598
 1599
 1600
 1601
 1602
 1603
 1604
 1605
 1606
 1607
 1608
 1609
 1610
 1611
 1612
 1613
 1614
 1615
 1616
 1617
 1618
 1619
 1620
 1621
 1622
 1623
 1624
 1625
 1626
 1627
 1628
 1629
 1630
 1631
 1632
 1633
 1634
 1635
 1636
 1637
 1638
 1639
 1640
 1641
 1642
 1643
 1644
 1645
 1646
 1647
 1648
 1649
 1650
 1651
 1652
 1653
 1654
 1655
 1656
 1657
 1658
 1659
 1660
 1661
 1662
 1663
 1664
 1665
 1666
 1667
 1668
 1669
 1670
 1671
 1672
 1673
 1674
 1675
 1676
 1677
 1678
 1679
 1680
 1681
 1682
 1683
 1684
 1685
 1686
 1687
 1688
 1689
 1690
 1691
 1692
 1693
 1694
 1695
 1696
 1697
 1698
 1699
 1700
 1701
 1702
 1703
 1704
 1705
 1706
 1707
 1708
 1709
 1710
 1711
 1712
 1713
 1714
 1715
 1716
 1717
 1718
 1719
 1720
 1721
 1722
 1723
 1724
 1725
 1726
 1727
 1728
 1729
 17210
 17211
 17212
 17213
 17214
 17215
 17216
 17217
 17218
 17219
 17220
 17221
 17222
 17223
 17224
 17225
 17226
 17227
 17228
 17229
 17230
 17231
 17232
 17233
 17234
 17235
 17236
 17237
 17238
 17239
 17240
 17241
 17242
 17243
 17244
 17245
 17246
 17247
 17248
 17249
 17250
 17251
 17252
 17253
 17254
 17255
 17256
 17257
 17258
 17259
 17260
 17261
 17262
 17263
 17264
 17265
 17266
 17267
 17268
 17269
 17270
 17271
 17272
 17273
 17274
 17275
 17276
 17277
 17278
 17279
 17280
 17281
 17282
 17283
 17284
 17285
 17286
 17287
 17288
 17289
 17290
 17291
 17292
 17293
 17294
 17295
 17296
 17297
 17298
 17299
 172100
 172101
 172102
 172103
 172104
 172105
 172106
 172107
 172108
 172109
 172110
 172111
 172112
 172113
 172114
 172115
 172116
 172117
 172118
 172119
 172120
 172121
 172122
 172123
 172124
 172125
 172126
 172127
 172128
 172129
 172130
 172131
 172132
 172133
 172134
 172135
 172136
 172137
 172138
 172139
 172140
 172141
 172142
 172143
 172144
 172145
 172146
 172147
 172148
 172149
 172150
 172151
 172152
 172153
 172154
 172155
 172156
 172157
 172158
 172159
 172160
 172161
 172162
 172163
 172164
 172165
 172166
 172167
 172168
 172169
 172170
 172171
 172172
 172173
 172174
 172175
 172176
 172177
 172178
 172179
 172180
 172181
 172182
 172183
 172184
 172185
 172186
 172187
 172188
 172189
 172190
 172191
 172192
 172193
 172194
 172195
 172196
 172197
 172198
 172199
 172200
 172201
 172202
 172203
 172204
 172205
 172206
 172207
 172208
 172209
 172210
 172211
 172212
 172213
 172214
 172215
 172216
 172217
 172218
 172219
 172220
 172221
 172222
 172223
 172224
 172225
 172226
 172227
 172228
 172229
 172230
 172231
 172232
 172233
 172234
 172235
 172236
 172237
 172238
 172239
 172240
 172241
 172242
 172243
 172244
 172245
 172246
 172247
 172248
 172249
 172250
 172251
 172252
 172253
 172254
 172255
 172256
 172257
 172258
 172259
 172260
 172261
 172262
 172263
 172264
 172265
 172266
 172267
 172268
 172269
 172270
 172271
 172272
 172273
 172274
 172275
 172276
 172277
 172278
 172279
 172280
 172281
 172282
 172283
 172284
 172285
 172286
 172287
 172288
 172289
 172290
 172291
 172292
 172293
 172294
 172295
 172296
 172297
 172298
 172299
 172300
 172301
 172302
 172303
 172304
 172305
 172306
 172307
 172308
 172309
 172310
 172311
 172312
 172313
 172314
 172315
 172316
 172317
 172318
 172319
 172320
 172321
 172322
 172323
 172324
 172325
 172326
 172327
 172328
 172329
 172330
 172331
 172332
 172333
 172334
 172335
 172336
 172337
 172338
 172339
 172340
 172341
 172342
 172343
 172344
 172345
 172346
 172347
 172348
 172349

A screenshot of a terminal window in a dark-themed IDE. The window displays Python code and its corresponding documentation annotations. The code includes a function that adds strings and floats. Annotations explain the behavior of the `add` operator for different types and the precision of floating-point numbers. The terminal also shows a file browser sidebar and a status bar at the bottom.

The screenshot shows a Jupyter Notebook environment with several open cells. The left sidebar lists files and notebooks, including 'calculator.ipynb' which is currently selected. The main area displays the following code:

```
312 if __name__ == "__main__":
313     pass
314
315 try:
316     result = add("5", 3)
317 except TypeError as e:
318     print(f"add('5', 3) raised {e.__class__.__name__}: {e}")
319
```

The code cell has a tooltip above it providing detailed documentation for the `calculator` module:

This module provides basic arithmetic operations which are comprehensive for document generation.
For documentation or automatic documentation generation using Python's built-in documentation tools (`help()`, `pydoc`, etc.).

The calculator module includes four classes:

- Addition
- Subtraction
- Multiplication
- Division

All functions include comprehensive docstrings that follow Google-style format.
One can be automatically generated and displayed using Python's documentation tools like `help()`, `pydoc`, and `epydoc`.

Possible Features:

- A type state for all parameters and return values.
- Comprehensive Google-style docstrings.
- Error handling with appropriate exceptions.
- Detailed examples in docstrings.
- Performance notes where applicable.

Usage:

```
>>> from calculator import add, subtract, multiply, divide
>>> add(5, 3)
8
>>> divide(10, 2)
5.0
>>> divide(10, 3)
Traceback (most recent call last):
...
ValueError: Cannot divide by zero
```

For module documentation, use:

```
>>> help(calculator)
>>> help(calculator.add)
>>> help(calculator.divide)
```

The status bar at the bottom indicates "In [1] Out[1]" and the date "23-05-2023".

The screenshot shows a Windows desktop environment with several open windows. In the foreground, a terminal window titled 'Calculator' displays Python code for a 'calculator' module. The code includes a main function that adds two numbers and handles a type error. Below the code, a 'PROBLEMS' section lists a single problem related to automatic documentation generation. A 'METHODS' section shows the 'help()' method being used to generate documentation for the module's functions, classes, and methods. The terminal also shows the output of the 'help()' command, which includes detailed descriptions for each item. To the right of the terminal, a file browser window is visible, showing a directory structure with files like 'calculator.py', 'calculator.pyc', and 'calculator.pdb'. The taskbar at the bottom shows icons for various applications, and the system tray indicates the date as 23-01-2018.

```
File Edit Selection View Go Run Terminal Help < > 312 if __name__ == "__main__":
313     print("\n--- Testing Error Handling ---\n")
314
315     try:
316         result = divide(10, 0)
317     except ValueError as e:
318         print(f"divide(10, 0) raised ValueError: {e}")
319
320     try:
321         result = add("5", 3)
322     except TypeError as e:
323         print(f"add('5', 3) raised TypeError: {e}")
324
325     print("\n" + "-" * 80)
326     print("Calculator module ready for documentation generation!")
327     print("-" * 80)
```

PROBLEMS: 1. AUTOMATIC DOCUMENTATION GENERATOR

INFO: 1: Using help() - Interactive Documentation

INFO: 2: Using help() - Interactive Documentation

INFO: 3: Using help() - Interactive Documentation

```
File Edit Selection View Go Run Terminal Help < > 312 if __name__ == "__main__":
313     print("\n--- Testing Basic Functionality ---\n")
314
315     print("add(10, 5) = " + str(add(10, 5)))
316     print("subtract(10, 3) = " + str(subtract(10, 3)))
317     print("multiply(4, 7) = " + str(multiply(4, 7)))
318     print("divide(20, 4) = " + str(divide(20, 4)))
319
320     print("\n--- Testing with Floating-Point Numbers ---\n")
321
322     print("add(3.5, 2.3) = " + str(add(3.5, 2.3)))
323     print("subtract(7.5, 2.3) = " + str(subtract(7.5, 2.3)))
324     print("multiply(2.5, 4) = " + str(multiply(2.5, 4)))
325     print("divide(7, 2) = " + str(divide(7, 2)))
326
327     print("\n--- Testing Edge Cases ---\n")
328
329     print("add(-5, 3) = " + str(add(-5, 3)))
330     print("multiply(-5, -3) = " + str(multiply(-5, -3)))
331     print("divide(0, 5) = " + str(divide(0, 5)))
332
333     print("\n--- Testing Error Handling ---\n")
334
335     print("add('5', 3) = " + str(add('5', 3)))
336     print("multiply('5', 3) = " + str(multiply('5', 3)))
337     print("divide(10, 0) = " + str(divide(10, 0)))
```

PROBLEMS: 1. AUTOMATIC DOCUMENTATION GENERATOR

INFO: 1: Using help() - Interactive Documentation

INFO: 2: Using help() - Interactive Documentation

INFO: 3: Using help() - Interactive Documentation

```
def divide(a: Union[int, float], b: Union[int, float]) -> float:
    See Also:
        - multiply(): Multiply two numbers
        - add(): Add two numbers
        - subtract(): Subtract two numbers
    """
    if not isinstance(a, (int, float)) or not isinstance(b, (int, float)):
        raise TypeError("Both a and b must be numeric types (int or float)")

    if b == 0:
        raise ValueError("Cannot divide by zero. The divisor must be non-zero.")

    return a / b

# DEMONSTRATION CODE (When run as main)
if __name__ == "__main__":
    print("= " * 80)
    print("CALCULATOR MODULE - DEMONSTRATION")
    print("= " * 80)
```

PROGRAM 1: AUTOMATIC DOCUMENTATION GENERATOR

```
def divide(a: Union[int, float], b: Union[int, float]) -> float:
    Examples:
        Division with remainder:
        >>> divide(7, 2)
        3.5

        Division by larger number (returns < 1):
        >>> divide(2, 5)
        0.4

        Division with negative numbers:
        >>> divide(-10, 2)
        -5.0

        Zero divided by a number:
        >>> divide(0, 5)
        0.0

        Division by zero raises error:
        >>> divide(10, 0)
        Traceback (most recent call last):
        ...
        ValueError: Cannot divide by zero
```

PROGRAM 1: AUTOMATIC DOCUMENTATION GENERATOR

RTFMOD 1: Using help() - Interactive Documentation

help() is Python's built-in documentation viewer. It displays docstrings in a formatted, readable way. To work from a terminal:

```
python -m pydoc calculator
```


A screenshot of a Jupyter Notebook interface. The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, Help, and a search bar. The left sidebar shows a file tree with various Python files like 'multiply.py', 'matrix.py', and 'string_operations.py'. The main notebook area contains a cell with the following code:

```
def multiply(a: Union[int, float], b: Union[int, float]) -> Union[int, float]:
```

Below the code, a tooltip provides detailed documentation for the 'multiply' function, including its parameters, return type, and examples. A status bar at the bottom shows 'File 1 of 1' and 'Running completed'. The bottom right corner displays system icons for battery, signal, and time.

```
113 def subtract(a: Union[int, float], b: Union[int, float]) -> Union[int, float]:  
114     """  
115     See Also:  
116         - add(): Add two numbers  
117         - multiply(): Multiply two numbers  
118         - divide(): Divide two numbers  
119     """  
120     if not isinstance(a, (int, float)) or not isinstance(b, (int, float)):  
121         raise TypeError("Both a and b must be numeric types (int or float)")  
122  
123     return a - b  
124  
125 def multiply(a: Union[int, float], b: Union[int, float]) -> Union[int, float]:  
126     """  
127     Multiply two numbers and return their product.  
128  
129     This function performs multiplication of two numeric values. It supports  
130     both integers and floating-point numbers. Multiplication is commutative,  
131     meaning a * b = b * a.  
132  
133     Args:  
134         a: First number (multiplicand). Can be positive, negative, or zero.  
135     """  
136     return a * b  
137  
138 def divide(a: Union[int, float], b: Union[int, float]) -> Union[int, float]:  
139     """  
140     Divide two numbers and return their quotient.  
141  
142     This function performs division of two numeric values. It supports  
143     both integers and floating-point numbers. Division is not commutative,  
144     meaning a / b != b / a.  
145  
146     Args:  
147         a: First number (dividend). Can be positive, negative, or zero.  
148         b: Second number (divisor). Must not be zero.  
149  
150     """  
151     if not isinstance(a, (int, float)) or not isinstance(b, (int, float)):  
152         raise TypeError("Both a and b must be numeric types (int or float)")  
153  
154     if b == 0:  
155         raise ZeroDivisionError("Cannot divide by zero")  
156  
157     return a / b  
158  
159 def mod(a: Union[int, float], b: Union[int, float]) -> Union[int, float]:  
160     """  
161     Get the remainder of a divided by b.  
162  
163     This function performs modulus operation of two numeric values. It supports  
164     both integers and floating-point numbers. Modulus is not commutative,  
165     meaning a % b != b % a.  
166  
167     Args:  
168         a: First number (dividend). Can be positive, negative, or zero.  
169         b: Second number (divisor). Must not be zero.  
170  
171     """  
172     if not isinstance(a, (int, float)) or not isinstance(b, (int, float)):  
173         raise TypeError("Both a and b must be numeric types (int or float)")  
174  
175     if b == 0:  
176         raise ZeroDivisionError("Cannot divide by zero")  
177  
178     return a % b  
179  
180 def power(a: Union[int, float], b: Union[int, float]) -> Union[int, float]:  
181     """  
182     Raise a to the power of b.  
183  
184     This function performs exponentiation of two numeric values. It supports  
185     both integers and floating-point numbers. Exponentiation is not commutative,  
186     meaning a ** b != b ** a.  
187  
188     Args:  
189         a: First number (base). Can be positive, negative, or zero.  
190         b: Second number (exponent). Must be an integer.  
191  
192     """  
193     if not isinstance(a, (int, float)) or not isinstance(b, int):  
194         raise TypeError("a must be numeric type (int or float) and b must be integer")  
195  
196     if b < 0:  
197         a = 1/a  
198  
199     result = 1  
200     for _ in range(b):  
201         result *= a  
202  
203     return result  
204  
205 if __name__ == "__main__":  
206     print(subtract(5, 3))  
207     print(multiply(4, 3))  
208     print(divide(10, 2))  
209     print(mod(10, 3))  
210     print(power(2, 3))  
211  
212 #!/usr/bin/python3  
213  
214 if __name__ == "__main__":  
215     print(subtract(5, 3))  
216     print(multiply(4, 3))  
217     print(divide(10, 2))  
218     print(mod(10, 3))  
219     print(power(2, 3))  
220  
221 if __name__ == "__main__":  
222     print(subtract(5, 3))  
223     print(multiply(4, 3))  
224     print(divide(10, 2))  
225     print(mod(10, 3))  
226     print(power(2, 3))  
227  
228 if __name__ == "__main__":  
229     print(subtract(5, 3))  
230     print(multiply(4, 3))  
231     print(divide(10, 2))  
232     print(mod(10, 3))  
233     print(power(2, 3))  
234  
235 if __name__ == "__main__":  
236     print(subtract(5, 3))  
237     print(multiply(4, 3))  
238     print(divide(10, 2))  
239     print(mod(10, 3))  
240     print(power(2, 3))  
241  
242 if __name__ == "__main__":  
243     print(subtract(5, 3))  
244     print(multiply(4, 3))  
245     print(divide(10, 2))  
246     print(mod(10, 3))  
247     print(power(2, 3))  
248  
249 if __name__ == "__main__":  
250     print(subtract(5, 3))  
251     print(multiply(4, 3))  
252     print(divide(10, 2))  
253     print(mod(10, 3))  
254     print(power(2, 3))  
255  
256 if __name__ == "__main__":  
257     print(subtract(5, 3))  
258     print(multiply(4, 3))  
259     print(divide(10, 2))  
260     print(mod(10, 3))  
261     print(power(2, 3))  
262  
263 if __name__ == "__main__":  
264     print(subtract(5, 3))  
265     print(multiply(4, 3))  
266     print(divide(10, 2))  
267     print(mod(10, 3))  
268     print(power(2, 3))  
269  
270 if __name__ == "__main__":  
271     print(subtract(5, 3))  
272     print(multiply(4, 3))  
273     print(divide(10, 2))  
274     print(mod(10, 3))  
275     print(power(2, 3))  
276  
277 if __name__ == "__main__":  
278     print(subtract(5, 3))  
279     print(multiply(4, 3))  
280     print(divide(10, 2))  
281     print(mod(10, 3))  
282     print(power(2, 3))  
283  
284 if __name__ == "__main__":  
285     print(subtract(5, 3))  
286     print(multiply(4, 3))  
287     print(divide(10, 2))  
288     print(mod(10, 3))  
289     print(power(2, 3))  
290  
291 if __name__ == "__main__":  
292     print(subtract(5, 3))  
293     print(multiply(4, 3))  
294     print(divide(10, 2))  
295     print(mod(10, 3))  
296     print(power(2, 3))  
297  
298 if __name__ == "__main__":  
299     print(subtract(5, 3))  
300     print(multiply(4, 3))  
301     print(divide(10, 2))  
302     print(mod(10, 3))  
303     print(power(2, 3))  
304  
305 if __name__ == "__main__":  
306     print(subtract(5, 3))  
307     print(multiply(4, 3))  
308     print(divide(10, 2))  
309     print(mod(10, 3))  
310     print(power(2, 3))  
311  
312 if __name__ == "__main__":  
313     print(subtract(5, 3))  
314     print(multiply(4, 3))  
315     print(divide(10, 2))  
316     print(mod(10, 3))  
317     print(power(2, 3))  
318  
319 if __name__ == "__main__":  
320     print(subtract(5, 3))  
321     print(multiply(4, 3))  
322     print(divide(10, 2))  
323     print(mod(10, 3))  
324     print(power(2, 3))  
325  
326 if __name__ == "__main__":  
327     print(subtract(5, 3))  
328     print(multiply(4, 3))  
329     print(divide(10, 2))  
330     print(mod(10, 3))  
331     print(power(2, 3))  
332  
333 if __name__ == "__main__":  
334     print(subtract(5, 3))  
335     print(multiply(4, 3))  
336     print(divide(10, 2))  
337     print(mod(10, 3))  
338     print(power(2, 3))  
339  
340 if __name__ == "__main__":  
341     print(subtract(5, 3))  
342     print(multiply(4, 3))  
343     print(divide(10, 2))  
344     print(mod(10, 3))  
345     print(power(2, 3))  
346  
347 if __name__ == "__main__":  
348     print(subtract(5, 3))  
349     print(multiply(4, 3))  
350     print(divide(10, 2))  
351     print(mod(10, 3))  
352     print(power(2, 3))  
353  
354 if __name__ == "__main__":  
355     print(subtract(5, 3))  
356     print(multiply(4, 3))  
357     print(divide(10, 2))  
358     print(mod(10, 3))  
359     print(power(2, 3))  
360  
361 if __name__ == "__main__":  
362     print(subtract(5, 3))  
363     print(multiply(4, 3))  
364     print(divide(10, 2))  
365     print(mod(10, 3))  
366     print(power(2, 3))  
367  
368 if __name__ == "__main__":  
369     print(subtract(5, 3))  
370     print(multiply(4, 3))  
371     print(divide(10, 2))  
372     print(mod(10, 3))  
373     print(power(2, 3))  
374  
375 if __name__ == "__main__":  
376     print(subtract(5, 3))  
377     print(multiply(4, 3))  
378     print(divide(10, 2))  
379     print(mod(10, 3))  
380     print(power(2, 3))  
381  
382 if __name__ == "__main__":  
383     print(subtract(5, 3))  
384     print(multiply(4, 3))  
385     print(divide(10, 2))  
386     print(mod(10, 3))  
387     print(power(2, 3))  
388  
389 if __name__ == "__main__":  
390     print(subtract(5, 3))  
391     print(multiply(4, 3))  
392     print(divide(10, 2))  
393     print(mod(10, 3))  
394     print(power(2, 3))  
395  
396 if __name__ == "__main__":  
397     print(subtract(5, 3))  
398     print(multiply(4, 3))  
399     print(divide(10, 2))  
400     print(mod(10, 3))  
401     print(power(2, 3))  
402  
403 if __name__ == "__main__":  
404     print(subtract(5, 3))  
405     print(multiply(4, 3))  
406     print(divide(10, 2))  
407     print(mod(10, 3))  
408     print(power(2, 3))  
409  
410 if __name__ == "__main__":  
411     print(subtract(5, 3))  
412     print(multiply(4, 3))  
413     print(divide(10, 2))  
414     print(mod(10, 3))  
415     print(power(2, 3))  
416  
417 if __name__ == "__main__":  
418     print(subtract(5, 3))  
419     print(multiply(4, 3))  
420     print(divide(10, 2))  
421     print(mod(10, 3))  
422     print(power(2, 3))  
423  
424 if __name__ == "__main__":  
425     print(subtract(5, 3))  
426     print(multiply(4, 3))  
427     print(divide(10, 2))  
428     print(mod(10, 3))  
429     print(power(2, 3))  
430  
431 if __name__ == "__main__":  
432     print(subtract(5, 3))  
433     print(multiply(4, 3))  
434     print(divide(10, 2))  
435     print(mod(10, 3))  
436     print(power(2, 3))  
437  
438 if __name__ == "__main__":  
439     print(subtract(5, 3))  
440     print(multiply(4, 3))  
441     print(divide(10, 2))  
442     print(mod(10, 3))  
443     print(power(2, 3))  
444  
445 if __name__ == "__main__":  
446     print(subtract(5, 3))  
447     print(multiply(4, 3))  
448     print(divide(10, 2))  
449     print(mod(10, 3))  
450     print(power(2, 3))  
451  
452 if __name__ == "__main__":  
453     print(subtract(5, 3))  
454     print(multiply(4, 3))  
455     print(divide(10, 2))  
456     print(mod(10, 3))  
457     print(power(2, 3))  
458  
459 if __name__ == "__main__":  
460     print(subtract(5, 3))  
461     print(multiply(4, 3))  
462     print(divide(10, 2))  
463     print(mod(10, 3))  
464     print(power(2, 3))  
465  
466 if __name__ == "__main__":  
467     print(subtract(5, 3))  
468     print(multiply(4, 3))  
469     print(divide(10, 2))  
470     print(mod(10, 3))  
471     print(power(2, 3))  
472  
473 if __name__ == "__main__":  
474     print(subtract(5, 3))  
475     print(multiply(4, 3))  
476     print(divide(10, 2))  
477     print(mod(10, 3))  
478     print(power(2, 3))  
479  
480 if __name__ == "__main__":  
481     print(subtract(5, 3))  
482     print(multiply(4, 3))  
483     print(divide(10, 2))  
484     print(mod(10, 3))  
485     print(power(2, 3))  
486  
487 if __name__ == "__main__":  
488     print(subtract(5, 3))  
489     print(multiply(4, 3))  
490     print(divide(10, 2))  
491     print(mod(10, 3))  
492     print(power(2, 3))  
493  
494 if __name__ == "__main__":  
495     print(subtract(5, 3))  
496     print(multiply(4, 3))  
497     print(divide(10, 2))  
498     print(mod(10, 3))  
499     print(power(2, 3))  
500  
501 if __name__ == "__main__":  
502     print(subtract(5, 3))  
503     print(multiply(4, 3))  
504     print(divide(10, 2))  
505     print(mod(10, 3))  
506     print(power(2, 3))  
507  
508 if __name__ == "__main__":  
509     print(subtract(5, 3))  
510     print(multiply(4, 3))  
511     print(divide(10, 2))  
512     print(mod(10, 3))  
513     print(power(2, 3))  
514  
515 if __name__ == "__main__":  
516     print(subtract(5, 3))  
517     print(multiply(4, 3))  
518     print(divide(10, 2))  
519     print(mod(10, 3))  
520     print(power(2, 3))  
521  
522 if __name__ == "__main__":  
523     print(subtract(5, 3))  
524     print(multiply(4, 3))  
525     print(divide(10, 2))  
526     print(mod(10, 3))  
527     print(power(2, 3))  
528  
529 if __name__ == "__main__":  
530     print(subtract(5, 3))  
531     print(multiply(4, 3))  
532     print(divide(10, 2))  
533     print(mod(10, 3))  
534     print(power(2, 3))  
535  
536 if __name__ == "__main__":  
537     print(subtract(5, 3))  
538     print(multiply(4, 3))  
539     print(divide(10, 2))  
540     print(mod(10, 3))  
541     print(power(2, 3))  
542  
543 if __name__ == "__main__":  
544     print(subtract(5, 3))  
545     print(multiply(4, 3))  
546     print(divide(10, 2))  
547     print(mod(10, 3))  
548     print(power(2, 3))  
549  
550 if __name__ == "__main__":  
551     print(subtract(5, 3))  
552     print(multiply(4, 3))  
553     print(divide(10, 2))  
554     print(mod(10, 3))  
555     print(power(2, 3))  
556  
557 if __name__ == "__main__":  
558     print(subtract(5, 3))  
559     print(multiply(4, 3))  
560     print(divide(10, 2))  
561     print(mod(10, 3))  
562     print(power(2, 3))  
563  
564 if __name__ == "__main__":  
565     print(subtract(5, 3))  
566     print(multiply(4, 3))  
567     print(divide(10, 2))  
568     print(mod(10, 3))  
569     print(power(2, 3))  
570  
571 if __name__ == "__main__":  
572     print(subtract(5, 3))  
573     print(multiply(4, 3))  
574     print(divide(10, 2))  
575     print(mod(10, 3))  
576     print(power(2, 3))  
577  
578 if __name__ == "__main__":  
579     print(subtract(5, 3))  
580     print(multiply(4, 3))  
581     print(divide(10, 2))  
582     print(mod(10, 3))  
583     print(power(2, 3))  
584  
585 if __name__ == "__main__":  
586     print(subtract(5, 3))  
587     print(multiply(4, 3))  
588     print(divide(10, 2))  
589     print(mod(10, 3))  
590     print(power(2, 3))  
591  
592 if __name__ == "__main__":  
593     print(subtract(5, 3))  
594     print(multiply(4, 3))  
595     print(divide(10, 2))  
596     print(mod(10, 3))  
597     print(power(2, 3))  
598  
599 if __name__ == "__main__":  
600     print(subtract(5, 3))  
601     print(multiply(4, 3))  
602     print(divide(10, 2))  
603     print(mod(10, 3))  
604     print(power(2, 3))  
605  
606 if __name__ == "__main__":  
607     print(subtract(5, 3))  
608     print(multiply(4, 3))  
609     print(divide(10, 2))  
610     print(mod(10, 3))  
611     print(power(2, 3))  
612  
613 if __name__ == "__main__":  
614     print(subtract(5, 3))  
615     print(multiply(4, 3))  
616     print(divide(10, 2))  
617     print(mod(10, 3))  
618     print(power(2, 3))  
619  
620 if __name__ == "__main__":  
621     print(subtract(5, 3))  
622     print(multiply(4, 3))  
623     print(divide(10, 2))  
624     print(mod(10, 3))  
625     print(power(2, 3))  
626  
627 if __name__ == "__main__":  
628     print(subtract(5, 3))  
629     print(multiply(4, 3))  
630     print(divide(10, 2))  
631     print(mod(10, 3))  
632     print(power(2, 3))  
633  
634 if __name__ == "__main__":  
635     print(subtract(5, 3))  
636     print(multiply(4, 3))  
637     print(divide(10, 2))  
638     print(mod(10, 3))  
639     print(power(2, 3))  
640  
641 if __name__ == "__main__":  
642     print(subtract(5, 3))  
643     print(multiply(4, 3))  
644     print(divide(10, 2))  
645     print(mod(10, 3))  
646     print(power(2, 3))  
647  
648 if __name__ == "__main__":  
649     print(subtract(5, 3))  
650     print(multiply(4, 3))  
651     print(divide(10, 2))  
652     print(mod(10, 3))  
653     print(power(2, 3))  
654  
655 if __name__ == "__main__":  
656     print(subtract(5, 3))  
657     print(multiply(4, 3))  
658     print(divide(10, 2))  
659     print(mod(10, 3))  
660     print(power(2, 3))  
661  
662 if __name__ == "__main__":  
663     print(subtract(5, 3))  
664     print(multiply(4, 3))  
665     print(divide(10, 2))  
666     print(mod(10, 3))  
667     print(power(2, 3))  
668  
669 if __name__ == "__main__":  
670     print(subtract(5, 3))  
671     print(multiply(4, 3))  
672     print(divide(10, 2))  
673     print(mod(10, 3))  
674     print(power(2, 3))  
675  
676 if __name__ == "__main__":  
677     print(subtract(5, 3))  
678     print(multiply(4, 3))  
679     print(divide(10, 2))  
680     print(mod(10, 3))  
681     print(power(2, 3))  
682  
683 if __name__ == "__main__":  
684     print(subtract(5, 3))  
685     print(multiply(4, 3))  
686     print(divide(10, 2))  
687     print(mod(10, 3))  
688     print(power(2, 3))  
689  
690 if __name__ == "__main__":  
691     print(subtract(5, 3))  
692     print(multiply(4, 3))  
693     print(divide(10, 2))  
694     print(mod(10, 3))  
695     print(power(2, 3))  
696  
697 if __name__ == "__main__":  
698     print(subtract(5, 3))  
699     print(multiply(4, 3))  
700     print(divide(10, 2))  
701     print(mod(10, 3))  
702     print(power(2, 3))  
703  
704 if __name__ == "__main__":  
705     print(subtract(5, 3))  
706     print(multiply(4, 3))  
707     print(divide(10, 2))  
708     print(mod(10, 3))  
709     print(power(2, 3))  
710  
711 if __name__ == "__main__":  
712     print(subtract(5, 3))  
713     print(multiply(4, 3))  
714     print(divide(10, 2))  
715     print(mod(10, 3))  
716     print(power(2, 3))  
717  
718 if __name__ == "__main__":  
719     print(subtract(5, 3))  
720     print(multiply(4, 3))  
721     print(divide(10, 2))  
722     print(mod(10, 3))  
723     print(power(2, 3))  
724  
725 if __name__ == "__main__":  
726     print(subtract(5, 3))  
727     print(multiply(4, 3))  
728     print(divide(10, 2))  
729     print(mod(10, 3))  
730     print(power(2, 3))  
731  
732 if __name__ == "__main__":  
733     print(subtract(5, 3))  
734     print(multiply(4, 3))  
735     print(divide(10, 2))  
736     print(mod(10, 3))  
737     print(power(2, 3))  
738  
739 if __name__ == "__main__":  
740     print(subtract(5, 3))  
741     print(multiply(4, 3))  
742     print(divide(10, 2))  
743     print(mod(10, 3))  
744     print(power(2, 3))  
745  
746 if __name__ == "__main__":  
747     print(subtract(5, 3))  
748     print(multiply(4, 3))  
749     print(divide(10, 2))  
750     print(mod(10, 3))  
751     print(power(2, 3))  
752  
753 if __name__ == "__main__":  
754     print(subtract(5, 3))  
755     print(multiply(4, 3))  
756     print(divide(10, 2))  
757     print(mod(10, 3))  
758     print(power(2, 3))  
759  
760 if __name__ == "__main__":  
761     print(subtract(5, 3))  
762     print(multiply(4, 3))  
763     print(divide(10, 2))  
764     print(mod(10, 3))  
765     print(power(2, 3))  
766  
767 if __name__ == "__main__":  
768     print(subtract(5, 3))  
769     print(multiply(4, 3))  
770     print(divide(10, 2))  
771     print(mod(10, 3))  
772     print(power(2, 3))  
773  
774 if __name__ == "__main__":  
775     print(subtract(5, 3))  
776     print(multiply(4, 3))  
777     print(divide(10, 2))  
778     print(mod(10, 3))  
779     print(power(2, 3))  
780  
781 if __name__ == "__main__":  
782     print(subtract(5, 3))  
783     print(multiply(4, 3))  
784     print(divide(10, 2))  
785     print(mod(10, 3))  
786     print(power(2, 3))  
787  
788 if __name__ == "__main__":  
789     print(subtract(5, 3))  
790     print(multiply(4, 3))  
791     print(divide(10, 2))  
792     print(mod(10, 3))  
793     print(power(2, 3))  
794  
795 if __name__ == "__main__":  
796     print(subtract(5, 3))  
797     print(multiply(4, 3))  
798     print(divide(10, 2))  
799     print(mod(10, 3))  
800     print(power(2, 3))  
801  
802 if __name__ == "__main__":  
803     print(subtract(5, 3))  
804     print(multiply(4, 3))  
805     print(divide(10, 2))  
806     print(mod(10, 3))  
807     print(power(2, 3))  
808  
809 if __name__ == "__main__":  
810     print(subtract(5, 3))  
811     print(multiply(4, 3))  
812     print(divide(10, 2))  
813     print(mod(10, 3))  
814     print(power(2, 3))  
815  
816 if __name__ == "__main__":  
817     print(subtract(5, 3))  
818     print(multiply(4, 3))  
819     print(divide(10, 2))  
820     print(mod(10, 3))  
821     print(power(2, 3))  
822  
823 if __name__ == "__main__":  
824     print(subtract(5, 3))  
825     print(multiply(4, 3))  
826     print(divide(10, 2))  
827     print(mod(10, 3))  
828     print(power(2, 3))  
829  
830 if __name__ == "__main__":  
831     print(subtract(5, 3))  
832     print(multiply(4, 3))  
833     print(divide(10, 2))  
834     print(mod(10, 3))  
835     print(power(2, 3))  
836  
837 if __name__ == "__main__":  
838     print(subtract(5, 3))  
839     print(multiply(4, 3))  
840     print(divide(10, 2))  
841     print(mod(10, 3))  
842     print(power(2, 3))  
843  
844 if __name__ == "__main__":  
845     print(subtract(5, 3))  
846     print(multiply(4, 3))  
847     print(divide(10, 2))  
848     print(mod(10, 3))  
849     print(power(2, 3))  
850  
851 if __name__ == "__main__":  
852     print(subtract(5, 3))  
853     print(multiply(4, 3))  
854     print(divide(10, 2))  
855     print(mod(10, 3))  
856     print(power(2, 3))  
857  
858 if __name__ == "__main__":  
859     print(subtract(5, 3))  
860     print(multiply(4, 3))  
861     print(divide(10, 2))  
862     print(mod(10, 3))  
863     print(power(2, 3))  
864  
865 if __name__ == "__main__":  
866     print(subtract(5, 3))  
867     print(multiply(4, 3))  
868     print(divide(10, 2))  
869     print(mod(10, 3))  
870     print(power(2, 3))  
871  
872 if __name__ == "__main__":  
873     print(subtract(5, 3))  
874     print(multiply(4, 3))  
875     print(divide(10, 2))  
876     print(mod(10, 3))  
877     print(power(2, 3))  
878  
879 if __name__ == "__main__":  
880     print(subtract(5, 3))  
881     print(multiply(4, 3))  
882     print(divide(10, 2))  
883     print(mod(10, 3))  
884     print(power(2, 3))  
885  
886 if __name__ == "__main__":  
887     print(subtract(5, 3))  
888     print(multiply(4, 3))  
889     print(divide(10, 2))  
890     print(mod(10, 3))  
891     print(power(2, 3))  
892  
893 if __name__ == "__main__":  
894     print(subtract(5, 3))  
895     print(multiply(4, 3))  
896     print(divide(10, 2))  
897     print(mod(10, 3))  
898     print(power(2, 3))  
899  
900 if __name__ == "__main__":  
901     print(subtract(5, 3))  
902     print(multiply(4, 3))  
903     print(divide(10, 2))  
904     print(mod(10, 3))  
905     print(power(2, 3))  
906  
907 if __name__ == "__main__":  
908     print(subtract(5, 3))  
909     print(multiply(4, 3))  
910     print(divide(10, 2))  
911     print(mod(10, 3))  
912     print(power(2, 3))  
913  
914 if __name__ == "__main__":  
915     print(subtract(5, 3))  
916     print(multiply(4, 3))  
917     print(divide(10, 2))  
918     print(mod(10, 3))  
919     print(power(2, 3))  
920  
921 if __name__ == "__main__":  
922     print(subtract(5, 3))  
923     print(multiply(4, 3))  
924     print(divide(10, 2))  
925     print(mod(10, 3))  
926     print(power(2, 3))  
927  
928 if __name__ == "__main__":  
929     print(subtract(5, 3))  
930     print(multiply(4, 3))  
931     print(divide(10, 2))  
932     print(mod(10, 3))  
933     print(power(2, 3))  
934  
935 if __name__ == "__main__":  
936     print(subtract(5, 3))  
937     print(multiply(4, 3))  
938     print(divide(10, 2))  
939     print(mod(10, 3))  
940     print(power(2, 3))  
941  
942 if __name__ == "__main__":  
943     print(subtract(5, 3))  
944     print(multiply(4, 3))  
945     print(divide(10, 2))  
946     print(mod(10, 3))  
947     print(power(2, 3))  
948  
949 if __name__ == "__main__":  
950     print(subtract(5, 3))  
951     print(multiply(4, 3))  
952     print(divide(10, 2))  
953     print(mod(10, 3))  
954     print(power(2, 3))  
955  
956 if __name__ == "__main__":  
957     print(subtract(5, 3))  
958     print(multiply(4, 3))  
959     print(divide(10, 2))  
960     print(mod(10, 3))  
961     print(power(2, 3))  
962  
963 if __name__ == "__main__":  
964     print(subtract(5, 3))  
965     print(multiply(4, 3))  
966     print(divide(10, 2))  
967     print(mod(10, 3))  
968     print(power(2, 3))  
969  
970 if __name__ == "__main__":  
971     print(subtract(5, 3))  
972     print(multiply(4, 3))  
973     print(divide(10, 2))  
974     print(mod(10, 3))  
975     print(power(2, 3))  
976  
977 if __name__ == "__main__":  
978     print(subtract(5, 3))  
979     print(multiply(4, 3))  
980     print(divide(10, 2))  
981     print(mod(10, 3))
```

Problem 4: Conversion Utilities Module

Task:

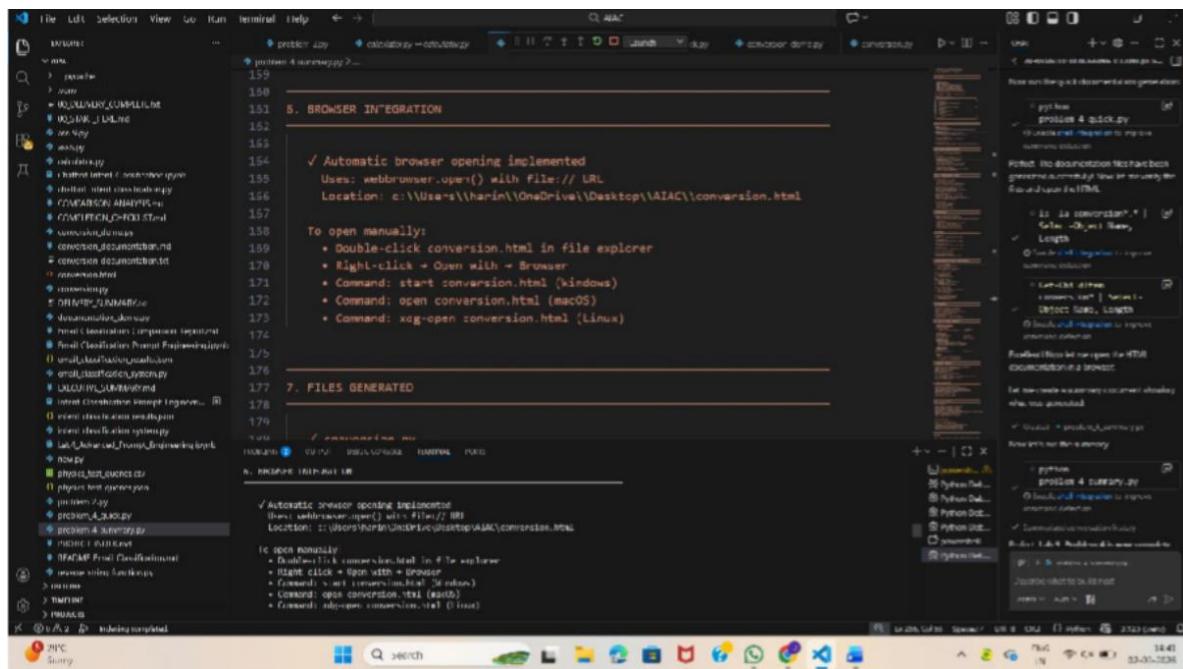
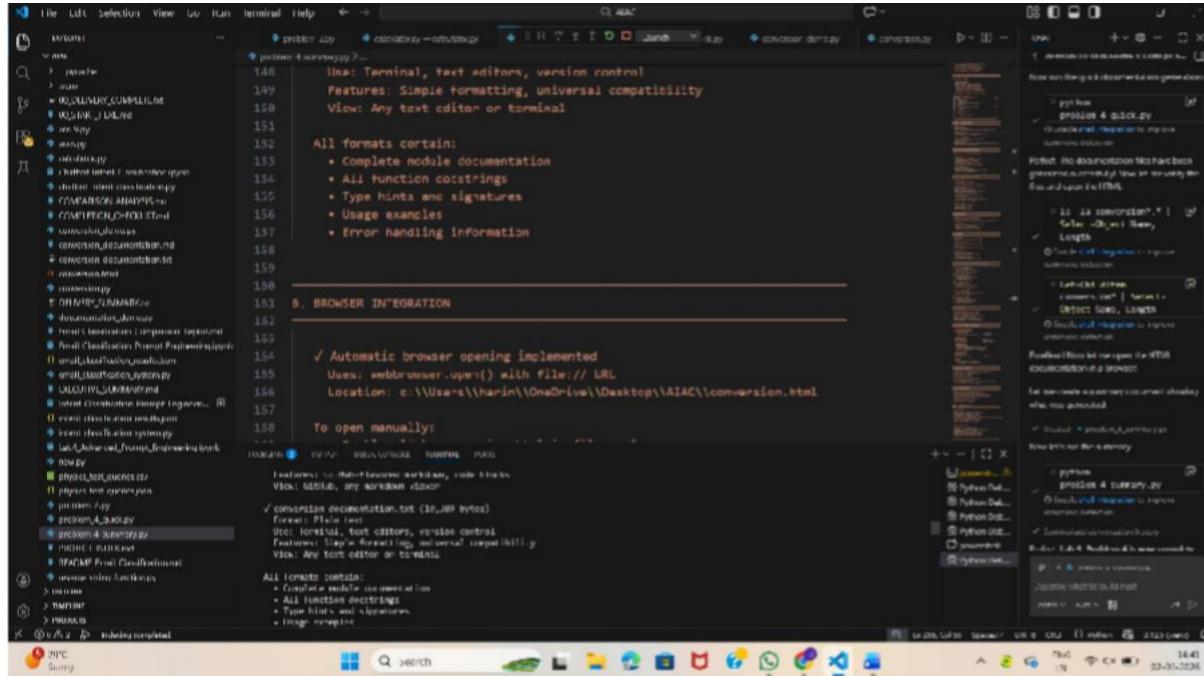
1. Write a module named conversion.py with functions:
 - o decimal_to_binary(n)
 - o binary_to_decimal(b)
 - o decimal_to_hexadecimal(n)

2. Use Copilot for auto-generating docstrings.

3. Generate documentation in the terminal.

4. Export the documentation in HTML format and open it in a browser.

Browser



```
File Edit Selection View Go Run Terminal Help < > problem_4.py conversion->conversion.py 379 'problem_4_quick.py': 'Quick generation script', 380 } 381 382 for filename, description in conversion_files.items(): 383 384 if os.path.exists(filename): 385 size = os.path.getsize(filename) 386 print(f'{filename:<35} {size:>8,} bytes - {description}') 387 print("\n" * 80) 388 print("PROBLEM 4 COMPLETE - ALL TASKS ACCOMPLISHED") 389 print("\n" * 80)
```

The terminal output shows the script running and printing the contents of the `conversion_files` dictionary. The output is as follows:

```
problem_4_quick.py: 'Quick generation script',
problem_4_main.py: 'Main module with main function'
conversion.py: 'Conversion module with conversion functions'
conversion.html: 'HTML documentation (open in browser)'
conversion_documentation.md: 'Markdown documentation'
conversion_documentation.html: 'Plain text documentation'
conversionrelated.html: 'Conversion related files'
conversion.py: 'Main module with conversion functions'
conversion.html: 'HTML documentation (open in browser)'
conversion_documentation.md: 'Markdown documentation'
conversion_documentation.html: 'Plain text documentation'
conversionrelated.html: 'Conversion related files'
conversion.py: 'Main module with conversion functions'
conversion.html: 'HTML documentation (open in browser)'
conversion_documentation.md: 'Markdown documentation'
conversion_documentation.html: 'Plain text documentation'
conversionrelated.html: 'Conversion related files'
conversion.py: 'Main module with conversion functions'
conversion.html: 'HTML documentation (open in browser)'
conversion_documentation.md: 'Markdown documentation'
conversion_documentation.html: 'Plain text documentation'
conversionrelated.html: 'Conversion related files'

PROBLEM 4 COMPLETE - ALL TASKS ACCOMPLISHED
```

```
File Edit Selection View Go Run Terminal Help < > problem_4.py conversion->conversion.py 379 'problem_4_quick.py': 'Quick generation script', 380 } 381 382 for filename, description in conversion_files.items(): 383 if os.path.exists(filename): 384 size = os.path.getsize(filename) 385 print(f'{filename:<35} {size:>8,} bytes - {description}') 386 print("\n" * 80) 387 print("PROBLEM 4 COMPLETE - ALL TASKS ACCOMPLISHED") 388 print("\n" * 80)
```

The terminal output shows the script running and printing the contents of the `conversion_files` dictionary. The output is as follows:

```
problem_4_quick.py: 'Quick generation script',
problem_4_main.py: 'Main module with main function'
conversion.py: 'Conversion module with conversion functions'
conversion.html: 'HTML documentation (open in browser)'
conversion_documentation.md: 'Markdown documentation'
conversion_documentation.html: 'Plain text documentation'
conversionrelated.html: 'Conversion related files'
conversion.py: 'Main module with conversion functions'
conversion.html: 'HTML documentation (open in browser)'
conversion_documentation.md: 'Markdown documentation'
conversion_documentation.html: 'Plain text documentation'
conversionrelated.html: 'Conversion related files'
conversion.py: 'Main module with conversion functions'
conversion.html: 'HTML documentation (open in browser)'
conversion_documentation.md: 'Markdown documentation'
conversion_documentation.html: 'Plain text documentation'
conversionrelated.html: 'Conversion related files'
conversion.py: 'Main module with conversion functions'
conversion.html: 'HTML documentation (open in browser)'
conversion_documentation.md: 'Markdown documentation'
conversion_documentation.html: 'Plain text documentation'
conversionrelated.html: 'Conversion related files'

PROBLEM 4 COMPLETE - ALL TASKS ACCOMPLISHED
```

```
problem_4_summary.py
379     'problem_4_quick.py': 'Quick generation script',
380
381
382     for filename, description in conversion_files.items():
383         if os.path.exists(filename):
384             size = os.path.getsize(filename)
385             print(f" {filename} ({size} bytes) - {description}")
386
387     print("\n" + "=" * 80)
388     print("PROBLEM 4 COMPLETE - ALL TASKS ACCOMPLISHED")
389     print("=" * 80)
390
391
392 CONCLUSION:
393 Problem 4 has been successfully completed. The conversion utilities module demonstrates professional-grade documentation practices:
394
395     • Source code documentation that automatically generates all outputs
396     • Multiple documentation delivery methods (Markdown, HTML, ReStructuredText)
397     • Professional HTML output suitable for websites
398     • Best practices in structuring artifacts
399     • Automatic documentation generation using Python's built-in tools
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
589 CONCLUSION:
590 Problem 4 has been successfully completed. The conversion utilities module demonstrates professional-grade documentation practices:
591
592     • Source code documentation that automatically generates all outputs
593     • Multiple documentation delivery methods (Markdown, HTML, ReStructuredText)
594     • Professional HTML output suitable for websites
595     • Best practices in structuring artifacts
596     • Automatic documentation generation using Python's built-in tools
```

```
File Edit Selection View Go Run Terminal Help + - ○ ABAZ
```

```
problem4_summary.py... 341 CONCLUSION:
342
343 Problem 4 has been successfully completed. The conversion utilities module
344 demonstrates professional-grade documentation practices:
345
346 • Source code documentation that automatically generates all formats
347 • Multiple documentation delivery methods (terminal, HTML, Markdown, text)
348 • Professional HTML output suitable for websites
349 • Best practices in coding writing
350 • Automatic documentation extraction using Python's built-in tools
351
352 The module is production-ready and demonstrates how proper documentation
353 enables automatic generation of multiple output formats from a single source.
354
355 Key Files:
356 • conversion.py - Main module (to use in your projects)
357 • conversion.html - HTML documentation (to share with others)
358 • conversion_demo.py - Complete demonstration script
359
360 • problem_4_quick.py - Quick generation script
```

```
CONCLUSION:
Problem 4 has been successfully completed. The conversion utilities module
demonstrates professional-grade documentation practices:
• Source code documentation that automatically generates all formats
• Multiple documentation delivery methods (terminal, HTML, Markdown, text)
• Professional HTML output suitable for websites
• Best practices in coding writing
• Automatic documentation extraction using Python's built-in tools
```

```
File Edit Selection View Go Run Terminal Help + - ○ ABAZ
```

```
File Edit Selection View Go Run Terminal Help + - ○ ABAZ
```

```
problem4_summary.py... 341 CONCLUSION:
342
343 Problem 4 has been successfully completed. The conversion utilities module
344 demonstrates professional-grade documentation practices:
345
346 • Implemented error handling with meaningful messages
347 • Added type hints to all functions
348 • Created comprehensive demonstration scripts
349 • Showed multiple documentation access methods
350 • Provided usage examples and best practices
351
352 CONCLUSION:
353
354 Problem 4 has been successfully completed. The conversion utilities module
355 demonstrates professional-grade documentation practices:
356
357 • Source code documentation that automatically generates all formats
358 • Multiple documentation delivery methods (terminal, HTML, Markdown, text)
359 • Professional HTML output suitable for websites
360 • Best practices in coding writing
361 • Automatic documentation extraction using Python's built-in tools
362
363 The module is production-ready and demonstrates how proper documentation
364 enables automatic generation of multiple output formats from a single source.
365
366 Additional Achievements:
367 • Generated Python documentation
368 • Generated HTML and documentation
369 • Implemented error handling with meaningful messages
```

```
CONCLUSION:
Problem 4 has been successfully completed. The conversion utilities module
demonstrates professional-grade documentation practices:
• Implemented error handling with meaningful messages
• Added type hints to all functions
• Created comprehensive demonstration scripts
• Showed multiple documentation access methods
• Provided usage examples and best practices
```

```
File Edit Selection View Go Run Terminal Help + - ○ ABAZ
```

PROBLEM 4: COMPLETION CHECKLIST

Task Requirements:

- Design a Python module named conversion.py
- Include decimal_to_binary(n) function
- Include binary_to_decimal(b) function
- Include decimal_to_hexadecimal(n) function
- Use Copilot for auto-generating docstrings ✓ (AI-assisted comprehensive docs)
- Generate documentation in the Terminal
- Export the documentation in HTML format
- Open the generated HTML in a browser

Additional Achievements:

- Generated Markdown documentation
- Generated Plain text documentation
- Implemented error handling with meaningful messages
- Added type hints to all functions

PROBLEM 4: COMPLETION CHECKLIST

Task Requirements:

- Design a Python module named conversion.py
- Include decimal_to_binary(n) function
- Include binary_to_decimal(b) function
- Include decimal_to_hexadecimal(n) function
- Use Copilot for auto-generating docstrings ✓ (AI-assisted comprehensive docs)
- Generate documentation in the Terminal
- Export the documentation in HTML format

12. TESTING AND VERIFICATION

- All functions tested with valid inputs
- Edge cases tested (0, power of 2, large numbers)
- Error handling verified (negative numbers, wrong types)
- Round-trip conversions verified (decimal-binary-decimal)
- Documentation generated successfully
- HTML files created and viewable
- Multiple export formats working
- Browser integration functional

PROBLEM 4: COMPLETION CHECKLIST

12. TESTING AND VERIFICATION

- All functions tested with valid inputs
- Edge cases tested (0, power of 2, large numbers)
- Error handling verified (negative numbers, wrong types)
- Round-trip conversions verified (decimal-binary-decimal)
- Documentation generated successfully
- HTML files created and viewable
- Multiple export formats working
- Browser integration functional

The image shows two side-by-side terminal windows on a Windows desktop, both running the same command to generate API documentation from a Python script.

Terminal 1 (Left):

```
python -m pydocx --conversion=conversion.py
```

Terminal 2 (Right):

```
python -m pydocx --conversion=conversion.py
```

In both terminals, the output is identical, displaying the generated API documentation:

```
11. BEST PRACTICES DEMONSTRATED
```

- ✓ Comprehensive docstrings (not just brief descriptions)
- ✓ Google-style format (industry standard)
- ✓ Type hints on all functions
- ✓ Multiple realistic examples
- ✓ Error documentation
- ✓ Performance notes (complexity analysis)
- ✓ "See Also" cross-references
- ✓ Machine-readable documentation (HTML generation works perfectly)
- ✓ Error handling with meaningful messages
- ✓ Module-level documentation

```
12. TESTING AND VERIFICATION
```

- ✓ Comprehensive docstrings (not just brief descriptions)
- ✓ Google-style format (industry standard)
- ✓ Type hints on all functions
- ✓ Multiple realistic examples
- ✓ Error documentation
- ✓ Performance notes (complexity analysis)
- ✓ "See Also" cross-references
- ✓ Machine-readable documentation (HTML generation works perfectly)
- ✓ Error handling with meaningful messages
- ✓ MultiTest documentation

```
11. API PRACTICES DEMONSTRATED
```

```
From Python Script:
```

```
import pydocx
pydocx.writedoc(conversion) # Generate HTML
```

```
With IDE:
```

- Hover over function name for tooltip
- Press Ctrl+Shift+F for documentation
- Use IDE's built-in documentation viewer

```
11. BEST PRACTICES DEMONSTRATED
```

- ✓ Comprehensive docstrings (not just brief descriptions)
- ✓ Google-style format (industry standard)
- ✓ Type hints on all functions
- ✓ Multiple realistic examples
- ✓ Error documentation

```
11. API PRACTICES DEMONSTRATED
```

```
import pydocx
pydocx.writedoc(conversion) # Generate HTML
```

```
With IDE:
```

- Hover over function name for tooltip
- Press F12/F11/F10 for documentation
- Use IDE's built-in documentation viewer

From Python Script:

```
import pydoc
pydoc.writedoc(conversion) # Generate HTML
```

With IDE:

- Hover over function name for tooltip
- Press Ctrl+Shift+I for documentation
- Use IDE's built-in documentation viewer

11. BEST PRACTICES DEMONSTRATED

- ✓ Comprehensive docstrings (not just brief descriptions)
- ✓ Google-style format (industry standard)
- ✓ Type hints on all functions
- ✓ Multiple realistic examples
- ✓ Error documentation

View Generated Files:

- Open conversion.html in any web browser
- Open conversion_documentation.md in GitHub/editor
- Open conversion_documentation.txt in any editor

From Python Shell:

```
import pydoc
pydoc.writedoc(conversion) # Generate HTML
```

View Generated Files:

- Open conversion.html in any web browser
- Open conversion_documentation.md in GitHub/editor
- Open conversion_documentation.txt in any editor

From Command Line:

```
python -m pydoc conversion # Terminal docs
pydoc -w conversion # Generate HTML
pydoc -b conversion # Open in browser
```

View Generated Files:

- Open conversion.html in any web browser
- Open conversion_documentation.md in GitHub/editor
- Open conversion_documentation.txt in any editor

12. HOW TO USE THE GENERATED DOCUMENTATION

In Python Interactive Shell:

```
>>> import conversion
>>> help(conversion) # Module help
>>> help(conversion.decimal_to_binary) # Function help
```

From Command Line:

```
python -m pydoc conversion # Terminal docs
pydoc -w conversion # Generate HTML
pydoc -b conversion # Open in browser
```

View Generated Files:

- Open conversion.html in any web browser
- Open conversion_documentation.md in GitHub/editor
- Open conversion_documentation.txt in any editor

13. HOW TO USE THE GENERATED DOCUMENTATION

In Python Interactive Shell:

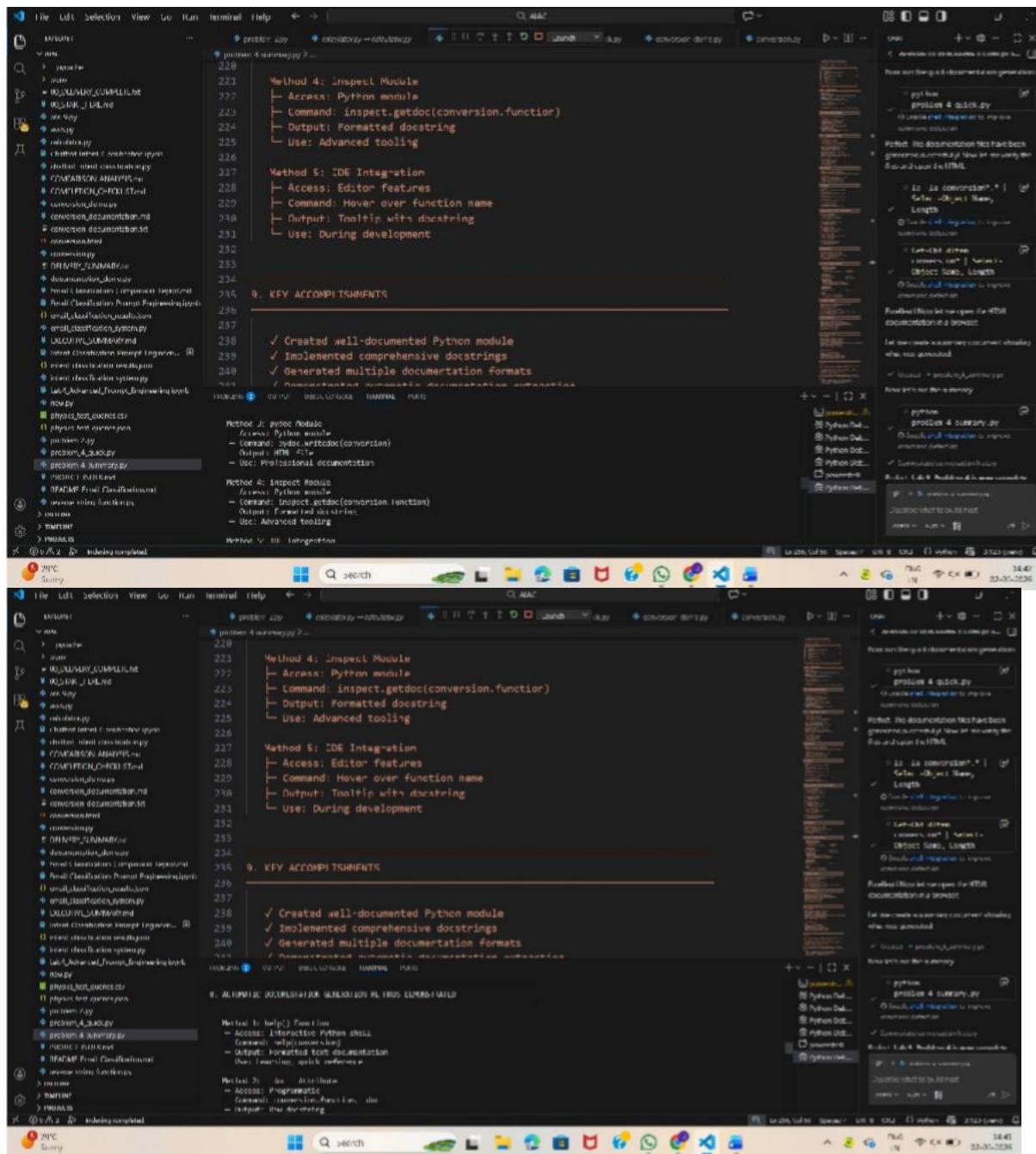
```
>>> import conversion
>>> help(conversion)
>>> help(conversion.decimal_to_binary) # Module help
>>> help(conversion.decimal_to_binary) # Function help
```

From Command Line:

```
python -m pydoc conversion # Terminal docs
pydoc -w conversion # Generate HTML
pydoc -b conversion # Open in browser
```

View Generated Files:

- Open conversion.html in any web browser
- Open conversion_documentation.md in GitHub/editor
- Open conversion_documentation.txt in any editor



A screenshot of a dual-monitor Python development environment. The left monitor displays a code editor with a tree view of files and a main text area showing code snippets for automatic documentation generation. The right monitor shows another code editor with a similar tree view and a main text area showing generated documentation. Both monitors have toolbars at the top and bottom, and a taskbar with various icons at the bottom.

Problem 5 – Course Management Module

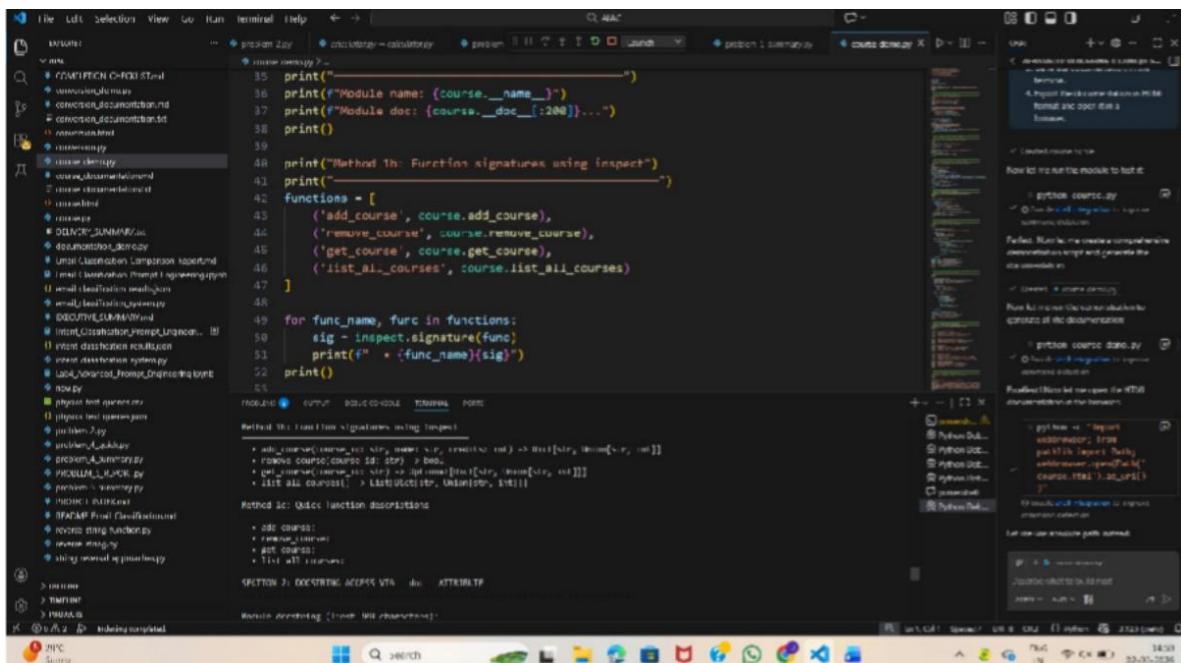
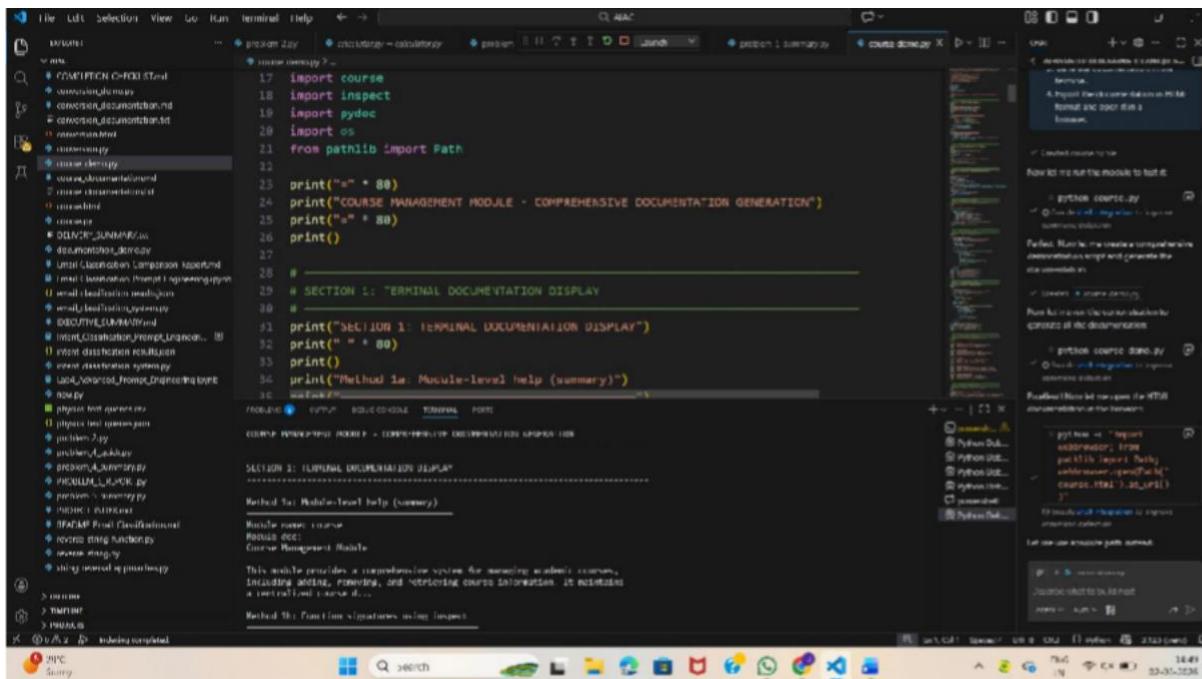
Task:

1. Create a module course.py with functions:
 - o add_course(course_id, name, credits)
 - o remove_course(course_id)
 - o get_course(course_id)

2. Add docstrings with Copilot.

3. Generate documentation in the terminal.

4. Export the documentation in HTML format and open it in a browser.



The screenshot displays two identical windows of a Python-based documentation generator, likely using Pygments for syntax highlighting. The application interface includes a top navigation bar with File, Edit, Selection, View, Go, Run, Terminal, Help, and a search bar. The main area contains code snippets for generating documentation, with sections like "SECTION 1: DOCUMENTATION GENERATION", "SECTION 2: COMPARISON OF DOCUMENTATION METHODS", "SECTION 3: VERIFYING GENERATED FILES", and "SECTION 7: BEST PRACTICES FOR DOCUMENTATION USAGE". Each section contains several print statements that output documentation details such as help(), inspect(), and pydoc module descriptions. The right pane of each window shows a file tree and a terminal-like interface with command history and execution results.

```
print("SECTION 1: DOCUMENTATION GENERATION")
print("  pydoc.writedoc(module) - Generate HTML")
print("  custom markdown export - For version control")
print("  Custom text export - For email/sharing")
print()

# SECTION 2: COMPARISON OF DOCUMENTATION METHODS
print("SECTION 2: COMPARISON OF DOCUMENTATION METHODS")
print("-" * 80)
print()

print("Method | Format | Use Case | Pros | Cons")
print("-" * 80)
print("help() | Terminal | Quick reference | Interactive, easy | Limited formatting")
print("_doc_ | Raw text | Programmatic access | Direct access | Unformatted")
print("inspect | Structured | Advanced tools | Detailed info | Requires code")
print("pydoc | HTML | Web | Professional | Beautiful, linked | Large file size")
print("Markdown | Text | Version control | Reusable, available | Manual export")
print("Plain text | Text | Universal | Simple, compatible | Basic formatting")
print()

# SECTION 3: VERIFYING GENERATED FILES
print("SECTION 3: VERIFYING GENERATED FILES")
print("-" * 80)
print()
print()

# SECTION 7: BEST PRACTICES FOR DOCUMENTATION USAGE
print("SECTION 7: BEST PRACTICES FOR DOCUMENTATION USAGE")
print("-" * 80)
print()

print("1. INTERACTIVE PYTHON SHELL")
print("  help(course) - Module documentation")
print("  help(course.add_course) - Function documentation")
print("  course.add_course.__doc__ - Raw docstring")
print()

print("2. IDE INTEGRATION")
print("  Hover over function name for tooltip")
print("  Ctrl+Shift+I (or Cmd+K,Cmd+I on Mac) for documentation")
print("  Intellisense shows docstring in suggestions")
print()

print("3. COMMAND LINE")
print("  python -m pydoc course - Terminal documentation")
print("  python -m pydoc -w course - Generate HTML")
print("  python -m pydoc -b course - Open in browser")
print()

print("4. PROGRAMMATIC ACCESS")
print("  import inspect")
print("  inspect.getdoc(course.add_course) - Formatted docstring")
print("  inspect.signature(course.add_course) - Function signature")
print("  inspect.getsource(course.add_course) - Function source code")
print()
```

The screenshot shows a code editor with two tabs open: 'course_docs.py' and 'course.html'. The 'course_docs.py' tab contains Python code for generating documentation from a module. The 'course.html' tab shows the generated HTML documentation.

```
def generate_text_docs(module):
    return text

    text_content = generate_text_docs(course)
    txt_path = Path(course_documentation.txt)
    txt_path.write_text(text_content, encoding='utf-8')
    print(f'Plain text documentation exported!')
    print(f' File: course_documentation.txt')
    print(f' Size: {txt_path.stat().st_size}, bytes')
    print()

# SECTION 7: BEST PRACTICES FOR DOCUMENTATION
print("SECTION 7: BEST PRACTICES FOR DOCUMENTATION USAGE")
print("-" * 80)
```

The 'course.html' tab displays the generated documentation:

PLAIN TEXT

- Python documentation created with comprehensive docstrings
- External documentation displayed (via help `help __doc__`)
- FILE documentation generated and saved
- Function documentation exported for review context
- Plain text documentation exported for universal access
- All files now documented with examples and error cases

HTML TIPS:

1. Open `course.html` in your web browser to view formatted docs
2. Click `course_documentation.html` for GitHub-friendly format
3. Use `python -m pydoc course` for terminal-based help
4. Report course module and use `help()` function for interactive docs

DOCUMENTATION GENERATOR OUTPUT:
D:\Users\steve\PycharmProjects\course\course>

This screenshot is identical to the one above, showing the same code editor interface with 'course_docs.py' and 'course.html' tabs open, displaying the same Python code and generated HTML documentation.

File Edit Selection View Go Run Terminal Help

Course Documentation

```

185     md_path.write_text(markdown_content, encoding='utf-8')
186     print(f'Course documentation exported!')
187     print(f' File: course_documentation.md')
188     print(f' Size: {md_path.stat().st_size,} bytes')
189     print()
190
191
192 # SECTION 6: PLAIN TEXT DOCUMENTATION EXPORT
193 #
194
195 print("SECTION 6: PLAIN TEXT DOCUMENTATION EXPORT")
196 print(" = 60")
197 print()
198
199 def generate_text_docs(module):
200     """Generate plain text documentation from module docstrings."""
201     text = f'{module.__name__.upper()} DOCUMENTATION\n'
202     for name, obj in inspect.getmembers(module, inspect.isfunction):
203         if not name.startswith('_'):
204             md += f'***** {name}\n'
205             sig = inspect.signature(obj)
206             md += f'{sig}\n{obj.__doc__}\n'
207             md += f'-----\n'
208
209     md += "\n\n"
210
211     return md
212
213 markdown_content = generate_markdown_docs(course)
214 md_path = Path('course_documentation.md')
215 md_path.write_text(markdown_content, encoding='utf-8')

```

PROBLEMS OUTPUT SOURCE CODE TERMINAL PORTS

SECTION 5: VERIFICATION OF GENERATED FILES

File	Description	Size
course.py	Main module	(16,742 bytes)
course.html	HTML documentation (open in browser)	(24,238 bytes)
course_documentation.html	Markdown documentation	(7,151 bytes)
course.documentation.txt	Plain text documentation	(11,836 bytes)

SECTION 6: SUMMARY AND RUNS FILES

DOCUMENTATION

- Foculus documentation created with comprehensive doctests
- Terminal documentation displayed (in help and --ix)
- Plain documentation generated and saved

SECTION 7: COMPARISON OF DOCUMENTATION METHODS

Retired | Forum | Use Case | Docs | Core

Help() | Imports | Quick Reference | Interactive, easy | Watched Competing
 IDEs | Run I/O | Persistent Session | Always active | Refactored
 Targets | Structure | Advanced tools | Detailed info | Requires root
 Python API | API | Professional | New Web Toolkit | Large File viewer
 Randomizer | Text | Version control | Localizable, portable | Always export
 Plain text | Text | Universal | Simple, compatible | Basic Processing

The screenshot shows a dual-monitor setup for Python documentation generation. Both monitors display the same code editor interface, likely PyCharm, with the following code open:

```
print("SECTION 5: MARKDOWN DOCUMENTATION EXPORT")
print("-" * 80)
print()

def generate_markdown_docs(module):
    """Generate Markdown documentation from module docstrings."""
    md = f"# {module.__name__.upper()} Documentation\n\n"

    if module.__doc__:
        md += f"## Module Overview\n\n"
        md += f"## {module.__doc__}\n\n"

    md += "## Functions\n\n"

    for name, obj in inspect.getmembers(module, inspect.isfunction):
        if inspect.isfunction(obj):
            md += f"### {name}\n"
            md += f"#### {obj.__name__} - {obj.__doc__}\n"
            md += f"##### {obj.__code__.co_name} - Hover over function name for tooltip\n"
            md += f"##### {obj.__code__.co_lnotab} - Line numbers for backtracking in Web for documented line\n"
            md += f"##### {obj.__code__.co_filename} - Intelligent sheet coding in suggestions\n"
            md += f"##### {obj.__code__.co_nlocals} - Local variables from the stack\n"
            md += f"##### {obj.__code__.co_stacksize} - Stack size\n"
            md += f"##### {obj.__code__.co_consts} - Constants\n"
            md += f"##### {obj.__code__.co_freevars} - Free variables\n"
            md += f"##### {obj.__code__.co_cellvars} - Cell variables\n"
            md += f"##### {obj.__code__.co_varnames} - Variable names\n"
            md += f"##### {obj.__code__.co_name} - Hover over variable name for tooltip\n"
            md += f"##### {obj.__code__.co_lnotab} - Line numbers for backtracking in Web for documented line\n"
            md += f"##### {obj.__code__.co_filename} - Intelligent sheet coding in suggestions\n"
            md += f"##### {obj.__code__.co_nlocals} - Local variables from the stack\n"
            md += f"##### {obj.__code__.co_stacksize} - Stack size\n"
            md += f"##### {obj.__code__.co_consts} - Constants\n"
            md += f"##### {obj.__code__.co_freevars} - Free variables\n"
            md += f"##### {obj.__code__.co_cellvars} - Cell variables\n            "
        else:
            md += f"### {name} - {obj.__doc__}\n"
            md += f"#### {obj.__name__} - {obj.__doc__}\n"
            md += f"##### {obj.__code__.co_name} - Hover over function name for tooltip\n"
            md += f"##### {obj.__code__.co_lnotab} - Line numbers for backtracking in Web for documented line\n"
            md += f"##### {obj.__code__.co_filename} - Intelligent sheet coding in suggestions\n"
            md += f"##### {obj.__code__.co_nlocals} - Local variables from the stack\n"
            md += f"##### {obj.__code__.co_stacksize} - Stack size\n"
            md += f"##### {obj.__code__.co_consts} - Constants\n"
            md += f"##### {obj.__code__.co_freevars} - Free variables\n"
            md += f"##### {obj.__code__.co_cellvars} - Cell variables\n            "
    return md
```

The code is part of a script named `generate_markdown_docs` which generates Markdown documentation from module docstrings. The code uses the `inspect` module to introspect functions and their docstrings, then formats them into a structured Markdown document.

The screenshot shows the PyCharm IDE interface with several windows open:

- Editor:** Displays Python code for generating course documentation. The code uses the `inspect` module to analyze function signatures and extract documentation strings.
- Toolbars:** Standard PyCharm toolbars for file operations, search, and navigation.
- Sidebar:** Shows project structure with files like `course.py`, `course.html`, and `course_directory.py`.
- Bottom Status Bar:** Shows the current file as `course_directory.py`, status `Working`, and a progress bar indicating the code is `Building completed`.

The main code area contains the following snippet:

```
def generate_summary():
    func = course.add_course
    sig = inspect.signature(func)
    print(f"Signature: {func.__name__}({sig})")
    print(f"Parameters:")
    for param_name, param in sig.parameters.items():
        print(f"  {param_name}: {param.annotation} if param.annotation != inspect.Parameter.empty else param.type")
    print(f"Return type: {sig.return_annotation} if sig.return_annotation != inspect.Signature.empty else None")
    print()
    print(f"Docstring summary:")
    docstring = inspect.getdoc(func)
    if docstring:
        first_para = docstring.split('\n\n')[0]
        print(f"  {first_para[:200]}...")
    print()
```

Below the code, there are two sections of generated documentation:

SECTION 4: HTML DOCUMENTATION GENERATION

```
Generating summary:  
  Add a new course to the course management database....
```

SECTION 4: HTML DOCUMENTATION GENERATION USING PARSER

```
Generating HTML documentation...  
  NOTE: COULD NOT  
  ✓ HTML FILE generated successfully!  
  ✓ CSS generated  
  ✓ JS generated  
  ✓ Images generated  
  ✓ Assets generated  
  ✓ Scripts generated  
  ✓ Location: C:\Users\harsh\OneDrive\Desktop\PyCharm\course.html
```

SECTION 5: PARSEDOC DOCUMENTATION EXPORT

```
✓ ParseDoc document successfully!  
  ✓ course_directory.html
```

The right side of the screen shows the `PyCharm Help` window with various documentation links.

The screenshot shows a Python development environment with several tabs open. The main code editor has the following content:

```
#!/usr/bin/python3
# generate_course_doc.py

# course_summary
#   print(course.__doc__[:300] + "...")

# print()
# print("Function docstring - add_course (first 400 characters):")
# print("-" * 88)
# if course.add_course.__doc__:
#     print(course.add_course.__doc__[:400] + ...)
# print()

# SECTION 3: USING INSPECT MODULE FOR PROGRAMMATIC ACCESS
# print("-" * 88)
# print()

print("Function: add_course")
print("-" * 88)
# already existing, it will be created as the new inheritance (overwritten).
# The function creates input parameters, and receives except one for needed
# inputs: course_id and doc...
SECTION 1: MANAGING POSSIBLE NON-PROGRAMMATIC ACCESS
```

Below the code editor, there is a detailed description of the `add_course` function:

Function: add_course

`Signature: add_course(course_id: str, name: str, credits: int) -> Dict[str, Any]`

Parameters:

- course_id: str
- name: str
- credits: int

Return type: typing.Dict[str, typing.Union[str, int]]

Documentation completed.

On the right side of the interface, there is a sidebar with various tools and options, including:

- File, Edit, Selection, View, Go, Run, terminal, Help menu.
- A search bar at the top right.
- A status bar at the bottom right showing system information like battery level (29%), CPU usage, and network speed.

The screenshot shows a Python development environment with multiple tabs open. The main code editor tab contains a script named `generate_docs.py` which prints function descriptions and access via the `__doc__` attribute. Below the code editor is a terminal window showing the output of running the script. A sidebar on the right displays various project files and a file browser.

```
print("Method 1: Quick function descriptions")
print("-----")
for func_name, func in functions:
    docstring = func.__doc__
    first_line = docstring.split("\n")[0] if docstring else "No documentation"
    print(f" {func_name}: {first_line}")
    print()

# -----
# SECTION 2: DOCSSTRING ACCESS VIA __doc__ ATTRIBUTE
#
print("SECTION 2: DOCSSTRING ACCESS VIA __doc__ ATTRIBUTE")
print("-" * 88)
print()
print("Module docstring (first 300 characters):")
print("-" * 88)
if course.__doc__:
```