

Assignment – 9.5

2303A51176

Task 1 – String Utilities Function

PROMPT :

```
#def reverse_string(text):
```

```
#return text[::-1]
```

#Write documentation in:(a) Docstring (b) Inline comments (c) Google-style documentation

CODE :

```
❶ AI-9.5.py > ...
1  #def reverse_string(text):
2  #return text[::-1]
3  #Write documentation in:(a) Docstring (b) Inline comments (c) Google-style documentation
4  # (a) Docstring
5  def reverse_string(text):
6      """
7          Reverses the given string.
8
9          Args:
10             text (str): The string to be reversed.
11
12         Returns:
13             str: The reversed string.
14             """
15     return text[::-1]
16  # (b) Inline comments
17  def reverse_string(text):
18      # Reverse the string using slicing
19      return text[::-1]
20  # (c) Google-style documentation
21  def reverse_string(text):
22      """
23          Reverses the given string.
24
25          Args:
26             text (str): The string to be reversed.
27
28          Returns:
29             str: The reversed string.
30             """
31     return text[::-1]
32  #Compare the three documentation styles.
33  # The docstring provides a clear and concise description of the function, its parameters, and its return value.
34  # It is a standard way to document functions in Python and can be easily accessed using the help() function.
35  # The inline comments provide a brief explanation of the code within the function.
36  # They are useful for explaining specific lines of code or logic but may not provide a comprehensive overview of the function's purpose and usage.
37  # The Google-style documentation is similar to the docstring but follows a specific format that is widely used in the industry.
38  # It includes sections for Args and Returns, making it easy to understand the function's inputs and outputs at a glance.
39  # This style is particularly beneficial for larger projects or when collaborating with other developers, as it provides a consistent and structured way to document code.
```

JUSTIFICATION :

Google-style documentation is most suitable for a utility-based string library because it clearly defines parameters and return values in a structured format. It improves readability and supports automatic documentation generation tools. It is widely used in professional development environments. Therefore, it enhances maintainability and scalability of utility libraries.

Task 2 – Password Strength Checker

PROMPT :

```
#def check_strength(password):
```

```
#return len(password) >= 8
```

#Write documentation in:(a) Docstring (b) Inline comments (c) Google-style documentation

CODE :

```
42 #def check_strength(password):
43     #return len(password) >= 8
44     #Write documentation in:(a) Docstring (b) Inline comments (c) Google-style documentation
45     # (a) Docstring
46     def check_strength(password):
47         """
48             Checks the strength of a password based on its length.
49
50             Args:
51                 password (str): The password to be checked.
52
53             Returns:
54                 bool: True if the password is strong (length >= 8), False otherwise.
55             """
56         return len(password) >= 8
57     # (b) Inline comments
58     def check_strength(password):
59         # Check if the length of the password is greater than or equal to 8
60         return len(password) >= 8
61     # (c) Google-style documentation
62     def check_strength(password):
63         """
64             Checks the strength of a password based on its length.
65
66             Args:
67                 password (str): The password to be checked.
68
69             Returns:
70                 bool: True if the password is strong (length >= 8), False otherwise.
71             """
72         return len(password) >= 8
73     #Compare the three documentation styles.
74     # The docstring provides a clear and concise description of the function, its parameters, and its return value.
75     # It is a standard way to document functions in Python and can be easily accessed using the help() function.
76     # The inline comments provide a brief explanation of the code within the function.
77     # They are useful for explaining specific lines of code or logic but may not provide a comprehensive overview of the function's purpose and usage.
78     # The Google-style documentation is similar to the docstring but follows a specific format that is widely
79     # used in the industry. It includes sections for Args and Returns, making it easy to understand the function's inputs and outputs at a glance.
80     # This style is particularly beneficial for larger projects or when collaborating with other developers, as it provides a consistent and structured way to document code.
```

JUSTIFICATION :

For security-related code, structured documentation like Google-style is preferred because it clearly explains validation logic and return behavior. Security functions require precise parameter and output descriptions to avoid misuse. Proper documentation also helps in auditing and maintenance. Hence, structured documentation improves reliability and clarity.

Task 3 – Math Utilities Module

PROMPT :

#Create a module math_utils.py with functions: square(n), cube(n), factorial(n)

#Write documentation in:(a) Docstring (b) Inline comments (c) Google-style documentation

CODE :

```

125 # (c) Google-style documentation
126 def square(n):
127     # Return the square of the input number
128     return n * n
129 def cube(n):
130     # Return the cube of the input number
131     return n * n * n
132 def factorial(n):
133     # Check if the input number is negative
134     if n < 0:
135         raise ValueError("Input must be a non-negative integer.")
136     # Return 1 for factorial of 0 and 1
137     elif n == 0 or n == 1:
138         return 1
139     else:
140         result = 1
141         # Calculate factorial using a loop
142         for i in range(2, n + 1):
143             result *= i
144         return result
145 # (c) Google-style documentation
146 def square(n):
147     """
148     Returns the square of a number.
149
150     Args:
151         n (int or float): The number to be squared.
152
153     Returns:
154         int or float: The square of the input number.
155     """
156     return n * n
157 def cube(n):
158     """
159     Returns the cube of a number.
160     Args:
161         n (int or float): The number to be cubed.
162     Returns:
163         int or float: The cube of the input number.
164     """
165     return n * n * n
166 def factorial(n):
167     """
168     Returns the factorial of a number.
169
170     #Create a module math_utils.py with functions: square(n), cube(n), factorial(n)
171     #Write documentation in:(a) Docstring (b) Inline comments (c) Google-style documentation
172     # math_utils.py
173     # (a) Docstring
174     def square(n):
175         """
176         Returns the square of a number.
177
178         Args:
179             n (int or float): The number to be squared.
180
181         Returns:
182             int or float: The square of the input number.
183         """
184         return n * n
185     def cube(n):
186         """
187         Returns the cube of a number.
188         Args:
189             n (int or float): The number to be cubed.
190         Returns:
191             int or float: The cube of the input number.
192         """
193         return n * n * n
194     def factorial(n):
195         """
196         Returns the factorial of a number.
197         Args:
198             n (int): The number to calculate the factorial of. Must be a non-negative integer.
199         Returns:
200             int: The factorial of the input number.
201         """
202         if n < 0:
203             raise ValueError("Input must be a non-negative integer.")
204         elif n == 0 or n == 1:
205             return 1
206         else:
207             result = 1
208             for i in range(2, n + 1):
209                 result *= i
210             return result
211 # (b) Inline comments

```

```

140     def square(n):
141         return n * n
142
143     def cube(n):
144         """
145             Returns the cube of a number.
146             Args:
147                 n (int or float): The number to be cubed.
148             Returns:
149                 int or float: The cube of the input number.
150         """
151         return n * n * n
152
153     def factorial(n):
154         """
155             Returns the factorial of a number.
156             Args:
157                 n (int): The number to calculate the factorial of. Must be a non-negative integer.
158             Returns:
159                 int: The factorial of the input number.
160         """
161         if n < 0:
162             raise ValueError("Input must be a non-negative integer.")
163         elif n == 0 or n == 1:
164             return 1
165         else:
166             result = 1
167             for i in range(2, n + 1):
168                 result *= i
169             return result
170
171     #Compare the three documentation styles.
172     # The docstring provides a clear and concise description of each function, its parameters, and its return value.
173     # It is a standard way to document functions in Python and can be easily accessed using the help() function.
174     # The inline comments provide a brief explanation of the code within each function.
175     # They are useful for explaining specific lines of code or logic but may not provide a comprehensive overview of the function's purpose and usage.
176     # The Google-style documentation is similar to the docstring but follows a specific format that is widely
177     # used in the industry. It includes sections for Args and Returns, making it easy to understand each function's inputs and outputs at a glance.
178     # This style is particularly beneficial for larger projects or when collaborating with other developers, as it provides a consistent and structured way to document code.
179
180

```

JUSTIFICATION :

Documenting mathematical functions improves clarity about input constraints and expected outputs. It helps prevent misuse, especially in functions like factorial where invalid inputs may cause errors. Automatically generated docstrings save time and maintain consistency. Exporting HTML documentation improves accessibility and professional presentation.

Task 4 – Attendance Management Module

PROMPT :

#Create a module attendance.py with functions: mark_present(student) ,
mark_absent(student) , get_attendance(student)

#Write documentation in:(a) Docstring (b) Inline comments (c) Google-style documentation

CODE :

```

237     return self.attendance_record.get(student, "No record found")
238 # (c) Google-style documentation
239 class Attendance:
240     """
241     A class to manage student attendance.
242
243     Methods:
244         mark_present(student): Marks a student as present.
245         mark_absent(student): Marks a student as absent.
246         get_attendance(student): Returns the attendance status of a student.
247
248     def __init__(self):
249         """Initializes an empty attendance record."""
250         self.attendance_record = {}
251
252     def mark_present(self, student):
253         """Marks a student as present.
254
255         Args:
256             student (str): The name of the student to be marked as present.
257             """
258             self.attendance_record[student] = "Present"
259
260     def mark_absent(self, student):
261         """Marks a student as absent.
262
263         Args:
264             student (str): The name of the student to be marked as absent.
265             """
266             self.attendance_record[student] = "Absent"
267
268     def get_attendance(self, student):
269         """Returns the attendance status of a student.
270
271         Args:
272             student (str): The name of the student whose attendance status is to be retrieved.
273
274         Returns:
275             str: The attendance status of the student, or a message if no record is found.
276             """
277             return self.attendance_record.get(student, "No record found")
278
279 #Compare the three documentation styles.
280 #The docstring provides a clear and concise description of the class and its methods, including their parameters and return values.

```

```

194 #Create a module attendance.py with functions: mark_present(student) , mark_absent(student) , get_attendance(student)
195 #Write documentation in:(a) Docstring (b) Inline comments (c) Google-style documentation
196 # attendance.py
197 # (a) Docstring
198 class Attendance:
199     """
200     A class to manage student attendance.
201
202     Methods:
203         mark_present(student): Marks a student as present.
204         mark_absent(student): Marks a student as absent.
205         get_attendance(student): Returns the attendance status of a student.
206
207     def __init__(self):
208         self.attendance_record = {}
209
210     def mark_present(self, student):
211         """Marks a student as present."""
212         self.attendance_record[student] = "Present"
213
214     def mark_absent(self, student):
215         """Marks a student as absent."""
216         self.attendance_record[student] = "Absent"
217
218     def get_attendance(self, student):
219         """Returns the attendance status of a student."""
220         return self.attendance_record.get(student, "No record found")
221
222 # (b) Inline comments
223 class Attendance:
224     def __init__(self):
225         # Initialize an empty dictionary to store attendance records
226         self.attendance_record = {}
227
228     def mark_present(self, student):
229         # Mark the student as present in the attendance record
230         self.attendance_record[student] = "Present"
231
232     def mark_absent(self, student):
233         # Mark the student as absent in the attendance record
234         self.attendance_record[student] = "Absent"
235
236     def get_attendance(self, student):
237         # Return the attendance status of the student, or a message if no record is found

```

```

420     self.attendance_record[student] = "Present"
259
260     def mark_absent(self, student):
261         """Marks a student as absent.
262
263         Args:
264             student (str): The name of the student to be marked as absent.
265
266             self.attendance_record[student] = "Absent"
267
268     def get_attendance(self, student):
269         """Returns the attendance status of a student.
270
271         Args:
272             student (str): The name of the student whose attendance status is to be retrieved.
273
274         Returns:
275             str: The attendance status of the student, or a message if no record is found.
276
277             return self.attendance_record.get(student, "No record found")
278
278 #Compare the three documentation styles.
279 # The docstring provides a clear and concise description of the class and its methods, including their parameters and return values.
280 # It is a standard way to document classes and methods in Python and can be easily accessed using the help() function.
281 # The inline comments provide a brief explanation of the code within each method.
282 # They are useful for explaining specific lines of code or logic but may not provide a comprehensive overview of the class's purpose and usage.
283 # The Google-style documentation is similar to the docstring but follows a specific format that is widely used in the industry. It includes sections for Args and Ret
284 # This style is particularly beneficial for larger projects or when collaborating with other developers, as it provides a consistent and structured way to document c

```

JUSTIFICATION :

Attendance systems require clear documentation to define function behavior and data handling. Proper docstrings improve maintainability and team collaboration. Generated documentation allows easy viewing in terminal and browser. This ensures better usability and transparency in academic or organizational systems.

Task 5 – File Handling Function

PROMPT :

```
#def read_file(filename):

#with open(filename, 'r') as f:

#return f.read()

#Write documentation in:(a) Docstring (b) Inline comments (c) Google-style documentation
```

CODE :

```

288     #def read_file(filename):
289     #with open(filename, 'r') as f:
290     #    return f.read()
291     #Write documentation in:(a) Docstring (b) Inline comments (c) Google-style documentation
292     # (a) Docstring
293     def read_file(filename):
294         """
295             Reads the contents of a file and returns it as a string.
296
297             Args:
298                 filename (str): The name of the file to be read.
299             Returns:
300                 str: The contents of the file.
301             """
302             with open(filename, 'r') as f:
303                 return f.read()
304             # (b) Inline comments
305             def read_file(filename):
306                 # Open the file in read mode
307                 with open(filename, 'r') as f:
308                     # Read the contents of the file and return it
309                     return f.read()
310             # (c) Google-style documentation
311             def read_file(filename):
312                 """
313                     Reads the contents of a file and returns it as a string.
314
315                     Args:
316                         filename (str): The name of the file to be read.
317                     Returns:
318                         str: The contents of the file.
319                     """
320                     with open(filename, 'r') as f:
321                         return f.read()
322             #Compare the three documentation styles.
323             # The docstring provides a clear and concise description of the function, its parameters, and its return value.
324             # It is a standard way to document functions in Python and can be easily accessed using the help() function.
325             # The inline comments provide a brief explanation of the code within the function.
326             # They are useful for explaining specific lines of code or logic but may not provide a comprehensive overview of the function's purpose and usage.
327             # The Google-style documentation is similar to the docstring but follows a specific format that is widely used in the industry.
328             # It includes sections for Args and Returns, making it easy to understand the function's inputs and outputs at a glance.
329             # This style is particularly beneficial for larger projects or when collaborating with other developers, as it provides a consistent and structured way to document code.

```

JUSTIFICATION :

Google-style documentation best explains exception handling because it clearly defines raised exceptions and return values. File operations are prone to errors like FileNotFoundError, so structured documentation improves understanding. It helps developers anticipate edge cases. Therefore, it enhances robustness and maintainability of file-handling code.