

ASSIGNMENT : 9.3

H.T.N.O : 2403A51292

BATCH : 12

Task :1

Basic Docstring Generation

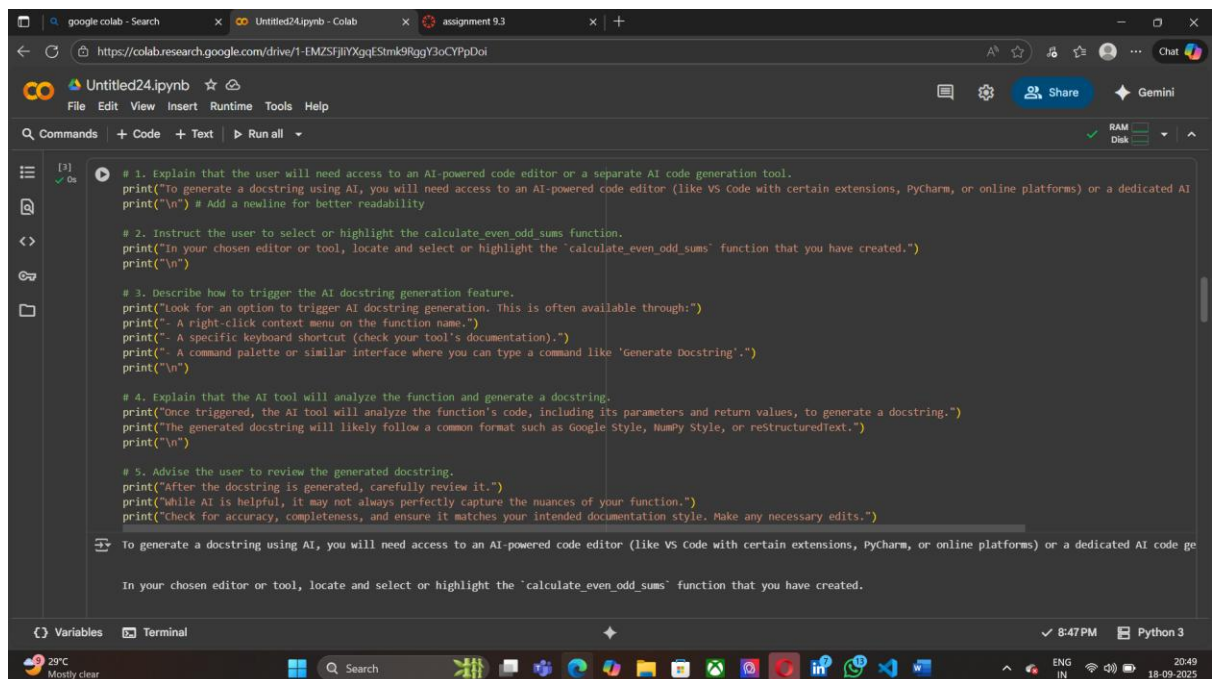
- Write python function to return sum of even and odd numbers in the given list.
- Incorporate manual docstring in code with Google Style
- Use an AI-assisted tool (e.g., Copilot, Cursor AI) to generate a docstring describing the function.
- Compare the AI-generated docstring with your manually written one

Expected Output#2: Students critically analyze AI-generated code comments.

Prompt:

Write a Python function that returns the sum of even and odd numbers in a given list.

- Manually add a Google-style docstring to the function describing its purpose, parameters, and return values.
- Use an AI-assisted tool (such as Copilot or Cursor AI) to generate a docstring for the same function.
- Compare the AI-generated docstring with your manually written one and note any differences in clarity, detail, or formatting.



```
1. Explain that the user will need access to an AI-powered code editor or a separate AI code generation tool.
print("To generate a docstring using AI, you will need access to an AI-powered code editor (like VS Code with certain extensions, PyCharm, or online platforms) or a dedicated AI code generation tool.")
print("\n") # Add a newline for better readability

2. Instruct the user to select or highlight the calculate_even_odd_sums function.
print("In your chosen editor or tool, locate and select or highlight the 'calculate_even_odd_sums' function that you have created.")
print("\n")

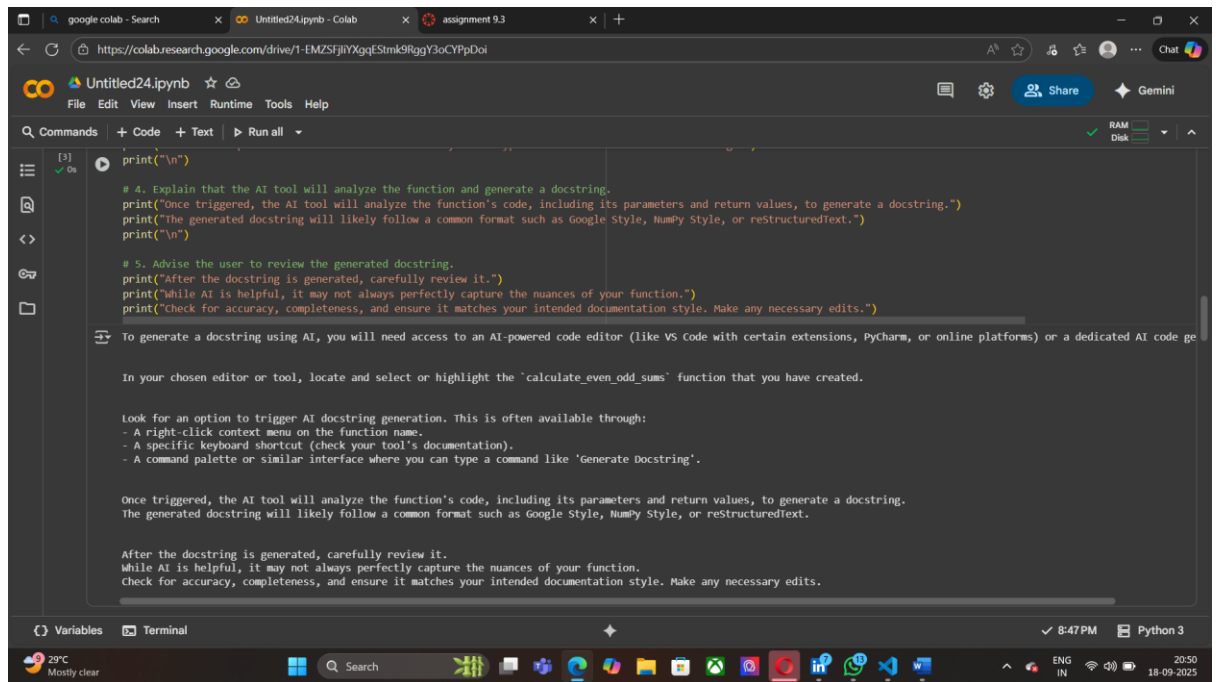
3. Describe how to trigger the AI docstring generation feature.
print("Look for an option to trigger AI docstring generation. This is often available through:")
print("- A right-click context menu on the function name.")
print("- A specific keyboard shortcut (check your tool's documentation).")
print("- A command palette or similar interface where you can type a command like 'Generate Docstring'.")
print("\n")

4. Explain that the AI tool will analyze the function and generate a docstring.
print("Once triggered, the AI tool will analyze the function's code, including its parameters and return values, to generate a docstring.")
print("The generated docstring will likely follow a common format such as Google style, NumPy style, or restructuredtext.")
print("\n")

5. Advise the user to review the generated docstring.
print("After the docstring is generated, carefully review it.")
print("While AI is helpful, it may not always perfectly capture the nuances of your function.")
print("Check for accuracy, completeness, and ensure it matches your intended documentation style. Make any necessary edits.")

To generate a docstring using AI, you will need access to an AI-powered code editor (like VS Code with certain extensions, PyCharm, or online platforms) or a dedicated AI code generation tool.

In your chosen editor or tool, locate and select or highlight the 'calculate_even_odd_sums' function that you have created.
```



TASK 2 :

Automatic Inline Comments

- Write python program for `sru_student` class with attributes like name, roll no., hostel_status and fee_update method and display_details method.
 - Write comments manually for each line/code block
 - Ask an AI tool to add inline comments explaining each line/step.
 - Compare the AI-generated comments with your manually written one.
- Expected Output#2: Students critically analyze AI-generated code comments.

Prompt:

Write a Python program defining an `sru_student` class with the following:

- Attributes: name, roll_no, hostel_status
 - Methods: fee_update and display_details
1. Manually add inline comments for each line or code block, explaining the logic.
 2. Use an AI tool (such as Copilot or Cursor AI) to generate inline comments for the same code.
 3. Compare the AI-generated comments with your manually written ones and analyze the differences in clarity, accuracy, and usefulness.

The screenshot shows a Google Colab notebook interface. At the top, there are browser tabs for 'google colab - Search', 'Untitled24.ipynb - Colab', and 'assignment 9.3'. The address bar shows a URL from colab.research.google.com. Below the browser tabs, there is a text prompt: 'Create an instance of the `sru_student` class and call the `display_details` method to ensure it works correct'. Below this, a 'Reasoning' section says: 'Create an instance of the `sru_student` class and call the `display_details` method to test it, as request'. The main code cell is labeled '[12]' and '0s'. It contains the following Python code:

```
# Create an instance of the sru_student class
student1 = sru_student(name="Alice Smith", roll_no="SRU12345", hostel_status="resident")

# Call the display_details method to display the student's information
student1.display_details()
```

The output of the code is displayed below the code cell:

```
--- Student Details ---
Name: Alice Smith
Roll Number: SRU12345
Hostel Status: resident
-----
```

Below the output, there is a 'Summary:' section and a 'Data Analysis Key Findings' section. At the bottom of the notebook, there is a status bar showing '29°C Mostly clear', a search bar, and various system icons including a clock showing '20:55 18-09-2025'.

Task 3 :

- Write a Python script with 3–4 functions (e.g., calculator: add, subtract, multiply, divide).
- Incorporate manual docstring in code with NumPy Style
- Use AI assistance to generate a module-level docstring + individual function docstrings.
- Compare the AI-generated docstring with your manually written one.

Expected Output#3: Students learn structured documentation for multi-function scripts

Prompt:

Write a Python script containing 3–4 functions (for example, a calculator with add, subtract, multiply, and divide functions).

1. Manually add NumPy-style docstrings to each function.
2. Use an AI tool (such as Copilot or Cursor AI) to generate a module-level docstring and individual function docstrings.
3. Compare the AI-generated docstrings with your manually written ones, noting differences in structure, clarity, and completeness.

The screenshot shows a Google Colab notebook titled 'Untitled24.ipynb'. The code cell contains a Python script with five numbered instructions for generating docstrings using an AI tool. The instructions are: 1. Explain the need for an AI-powered code editor. 2. Instruct the user to open a Python script with calculator functions. 3. Describe how to trigger the AI docstring generation feature. 4. Explain that the AI tool will analyze the code and generate docstrings. 5. Advise the user to carefully review the AI-generated docstrings. The script uses print statements to output these instructions. The Colab interface includes a menu bar (File, Edit, View, Insert, Runtime, Tools, Help), a toolbar with icons for commands, code, text, and running all code, and a status bar at the bottom showing the time as 8:58 PM and the environment as Python 3.

```
[17] ✓ 0s
# 1. Explain that the user will need access to an AI-powered code editor or a separate AI code generation tool.
print("To generate docstrings using an AI tool, you will need access to an AI-powered code editor (like VS Code with certain extensions, PyCharm, or online platforms that offer")
print("\n") # Add a newline for better readability

# 2. Instruct the user to open the Python script containing the calculator functions in their chosen environment.
print("Open the Python script containing the 'add', 'subtract', 'multiply', and 'divide' functions in your chosen development environment or AI tool.")
print("\n") # Add a newline for better readability

# 3. Describe how to trigger the AI docstring generation feature.
print("To generate docstrings:")
print("- For a module-level docstring, some tools might offer an option when the file is open or when selecting the entire file content. Look for options like 'Generate Module Docstring'")
print("- For individual function docstrings, select or place your cursor within the 'add', 'subtract', 'multiply', and 'divide' function definitions one by one. Then, look for options like 'Generate Function Docstring'")
print("These options are typically found in a right-click context menu on the code, within an 'AI' or 'Code Actions' menu, or via a dedicated command palette.")
print("\n") # Add a newline for better readability

# 4. Explain that the AI tool will analyze the code and generate docstrings.
print("Once triggered, the AI tool will analyze the code. For the module, it will look at the overall content. For each function, it will analyze the function signature (name, parameters, return values, and exceptions).")
print("The AI will then generate docstrings, often attempting to follow common styles like NumPy, Google, or reStructuredText, based on its training data or your tool's settings.")
print("\n") # Add a newline for better readability

# 5. Advise the user to carefully review the AI-generated docstrings.
print("After the AI generates the docstrings, it is crucial to carefully review them.")
print("Check for:")
print("- Accuracy: Does the docstring correctly describe what the code does?")
print("- Completeness: Does it include all parameters, return values, and exceptions?")
print("- Style: Does it adhere to the desired style (NumPy style in this case)?")
print("Make any necessary edits to ensure the docstrings are accurate, complete, and consistent with your project's documentation standards.")

To generate docstrings using an AI tool, you will need access to an AI-powered code editor (like VS Code with certain extensions, PyCharm, or online platforms that offer AI coding assistance).
```

The screenshot shows the same Google Colab notebook, but now the code cell has been executed. The output of the script is displayed in the cell, showing the five numbered instructions for generating docstrings using an AI tool. The output is formatted with line numbers and status indicators. The Colab interface is the same as in the first screenshot, with the same menu bar, toolbar, and status bar.

```
[17] ✓ 0s
# 5. Advise the user to carefully review the AI-generated docstrings.
print("After the AI generates the docstrings, it is crucial to carefully review them.")
print("Check for:")
print("- Accuracy: Does the docstring correctly describe what the code does?")
print("- Completeness: Does it include all parameters, return values, and exceptions?")
print("- Style: Does it adhere to the desired style (NumPy style in this case)?")
print("Make any necessary edits to ensure the docstrings are accurate, complete, and consistent with your project's documentation standards.")

To generate docstrings using an AI tool, you will need access to an AI-powered code editor (like VS Code with certain extensions, PyCharm, or online platforms that offer AI coding assistance).

Open the Python script containing the 'add', 'subtract', 'multiply', and 'divide' functions in your chosen development environment or AI tool.

To generate docstrings:
- For a module-level docstring, some tools might offer an option when the file is open or when selecting the entire file content. Look for options like 'Generate Module Docstring'
- For individual function docstrings, select or place your cursor within the 'add', 'subtract', 'multiply', and 'divide' function definitions one by one. Then, look for options like 'Generate Function Docstring'
These options are typically found in a right-click context menu on the code, within an 'AI' or 'Code Actions' menu, or via a dedicated command palette.

Once triggered, the AI tool will analyze the code. For the module, it will look at the overall content. For each function, it will analyze the function signature (name, parameter, return values, and exceptions).
The AI will then generate docstrings, often attempting to follow common styles like NumPy, Google, or reStructuredText, based on its training data or your tool's settings.

After the AI generates the docstrings, it is crucial to carefully review them.
Check for:
- Accuracy: Does the docstring correctly describe what the code does?
- Completeness: Does it include all parameters, return values, and exceptions?
- Style: Does it adhere to the desired style (NumPy style in this case)?
Make any necessary edits to ensure the docstrings are accurate, complete, and consistent with your project's documentation standards.
```