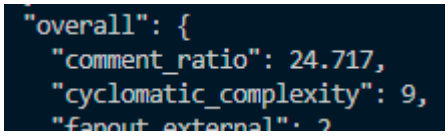
Link del repositorio: https://github.com/mirkojp/TP1/tree/main/src/chatGPT

1) Ejecute el programa multimetric sobre la última versión que disponga del programa utilizado en éste practico. Realice las siguientes consignas:

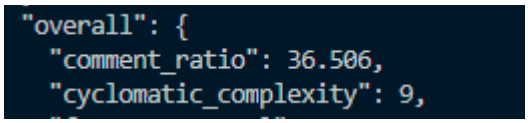
a. De la sección “Overall” del resultado verifique el resultado de “comment_ratio” (proporción de comentarios). Si éste es menor de 1/3 (33%) explore medidas para mejorar éste parámetro hasta un valor en ese entorno.
b. Explore el significado de y luego compare los valores de halstead_effort y halstead_timerequired con los que efectivamente le tomó el programa.
c. Como compara el valor halstead_bugprop con la cantidad de defectos que tuvo que solucionar luego que lograra que el programa ejecute por primer vez (es decir, excluyendo errores de sintaxis).
d. Que estrategias cree que se pueden aplicar para reducir el índice de McCabe (cyclomatic_complexity) en un 10%.
e. Capture en la entrega del práctico las distintas acciones y resultados que va obteniendo

a.

Cantidad Inicial de comentarios



Cantidad final de comentarios



b.

Halstead Effort (Esfuerzo de Halstead):

Esta métrica representa la cantidad de trabajo requerido para implementar un programa. Se calcula utilizando la fórmula:

$$E = V \cdot \log_2 N$$

Donde:

- *E* es el esfuerzo de Halstead.
- *V* es el vocabulario del programa, es decir, el número total de operadores y operando distintos en el programa.
- *N* es el tamaño del programa, es decir, el número total de operadores y operando en el programa

Halstead Time Required (Tiempo Requerido de Halstead):

Esta métrica representa el tiempo estimado necesario para implementar el programa. Se calcula utilizando la fórmula:

$$T = \frac{E}{18}$$

Donde:

- *T* es el tiempo requerido de Halstead.
- *E* es el esfuerzo de Halstead.

- **halstead_effort**: 67677.695
- **halstead_timerequired**: 3759.872≈1.04497hs

Desarrollar el programa tardo 3h, halstead_timerequired, subestima el tiempo que podría tomar en desarrollar el programa

c.

Iniciamos con una halstead_bugprop de 0.803

Solucionamos los siguientes errores:

- Cambiar la librería openai por OpenAI: Se actualiza la importación para utilizar la librería OpenAI en lugar de la versión anterior no soportada.

- Definir placeholders para context y usertask: Se definen variables para context y usertask como placeholders dentro de la función chat_with_gpt.

- Añadir librería argparse para gestionar la entrada de la línea de comandos: Se importa la librería argparse para poder gestionar las opciones de línea de comandos, lo que permitirá activar o desactivar el modo de conversación.

- Añadir condicional a la función chat_with_gpt: Se añade un condicional en la función chat_with_gpt para ejecutar distintas acciones dependiendo de si se trata de una conversación o no. Se modifican las variables adecuadamente en función de si es una conversación o no.

Al solucionar estos errores la halstead_bugprop es de 0.7, lo que podría significar que aún quedan muchos errores de lógica sin solucionar

d.

Para reducir el índice de complejidad ciclomática en este código (De valor actual 9), podemos

- División de la función principal main(): La función main() es bastante larga y realiza múltiples tareas. Podemos dividirla en funciones más pequeñas para manejar diferentes aspectos del programa, como Main y modo de conversación

Así conseguir reducirla a 8

Codigo inicial

```python
def main():

    # Global variable declaration to access the conversation history
    global conversation_history

    # Argument parsing for conversation mode
    # This line creates an ArgumentParser object,
    # which is used to define what command-line arguments the script should accept.
    parser = argparse.ArgumentParser(description="ChatGPT Conversational Mode")

    # Flag. When this flag is provided in the command line, it sets the value of args.convers
    # to True. The action="store_true" parameter indicates that if the --convers flag is present
    # , the value stored for this argument will be True
    parser.add_argument(
        "--convers", action="store_true", help="Activate conversation mode"
    )

    # This line parses the command-line arguments provided by the user using the parser defined above
    args = parser.parse_args()

    # Check if conversation mode is activated
    if args.convers:
        print("Modo de conversación activado.")

        # Conversation mode loop
        while True:
            try:
                # Prompt user for input
                user_query = input("Ingrese su consulta: ")
```

```python
# Check if conversation mode is activated
if args.convers:
    print("Modo de conversación activado.")

    # Conversation mode loop
    while True:
        try:
            # Prompt user for input
            user_query = input("Ingrese su consulta: ")

            # Check if user input is empty
            if user_query.strip() == "":
                print("Por favor, ingrese una consulta válida.")
                continue

            print("You:", user_query)

            # Call chat_with_gpt function with conversation mode activated
            response = chat_with_gpt(user_query, 1)
            if response is not None:
                print("chatGPT:", response)
        except KeyboardInterrupt:
            print("\nSaliendo del programa.")
            break
        except Exception as e:
            print("Error:", e)

else:
```

```
    else:
        # If conversation mode is not activated
        print("No se ha activado el modo de conversación.")

        # Non-conversation mode loop
        while True:
            try:
                # Prompt user for input
                user_query = input("Ingrese su consulta: ")

                # Check if user input is empty
                if user_query.strip() == "":
                    print("Por favor, ingrese una consulta válida.")
                    continue

                print("You:", user_query)

                # Call chat_with_gpt function with conversation mode deactivated
                response = chat_with_gpt(user_query, 0)
                if response is not None:
                    print("chatGPT:", response)
            except KeyboardInterrupt:
                print("\nSaliendo del programa.")
                break
            except Exception as e:
                print("Error:", e)
```

Codigo final

```
# Función para manejar la interacción con el usuario y llamar a la función chat_with_gpt
def handle_user_interaction(conversation):
    global conversation_history
    while True:
        try:
            user_query = input("Ingrese su consulta: ")
            if user_query.strip() == "":
                print("Por favor, ingrese una consulta válida.")
                continue
            print("You:", user_query)
            response = chat_with_gpt(user_query, conversation)
            if response is not None:
                print("chatGPT:", response)
        except KeyboardInterrupt:
            print("\nSaliendo del programa.")
            break
        except Exception as e:
            print("Error:", e)
```

```python
# Función principal que maneja la lógica de conversación
def main():
    global conversation_history
    parser = argparse.ArgumentParser(description="ChatGPT Conversational Mode")
    parser.add_argument(
        "--convers", action="store_true", help="Activate conversation mode"
    )
    args = parser.parse_args()

    if args.convers:
        print("Modo de conversación activado.")
        handle_user_interaction(1)
    else:
        print("No se ha activado el modo de conversación.")
        handle_user_interaction(0)


if __name__ == "__main__":
    main()
```

2) Instale un analizador estático de código denominado pylint (mediante el comando pip install pylint) y realice las siguientes acciones.
    a. Ejecute el programa pylint sobre el programa python desarrollado como parte de las consignas del punto -4- anterior.
    b. Analice el resultado que arroja. i. Introduzca correcciones en el programa fuente para abordar los comentarios. Incluya los comentarios en la primer corrida y como queda luego de todas las correcciones realizadas.
    c. Aquellos comentarios que decida no abordar justifique brevemente porque decidió ignorar las recomendaciones.

a.

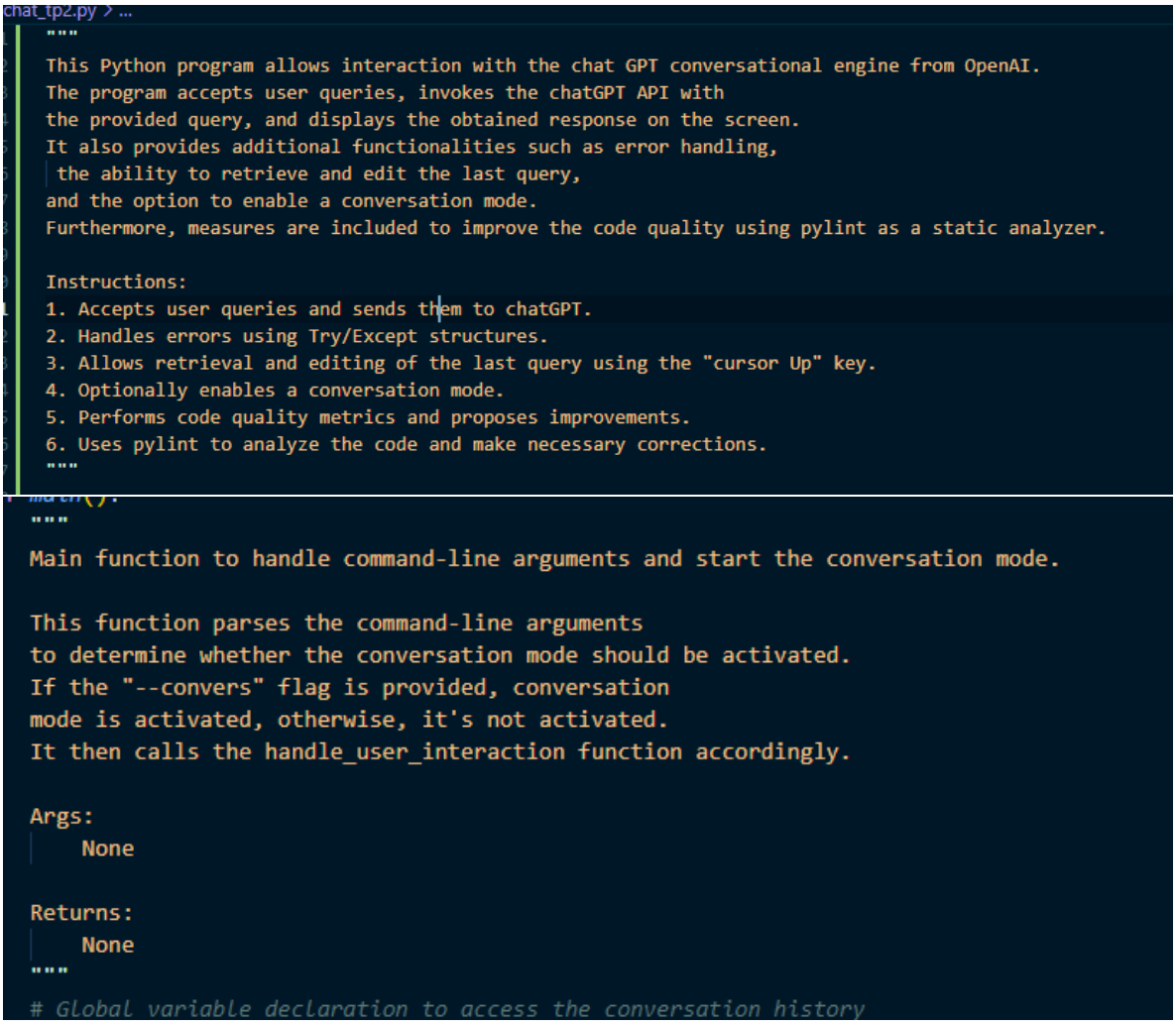
```
PS C:\Users\Admin\Desktop\Ing en software\IS2\TP2> pylint Chat_TP2.py
************* Module Chat_TP2
Chat_TP2.py:85:0: C0301: Line too long (101/100) (line-too-long)
Chat_TP2.py:1:0: C0114: Missing module docstring (missing-module-docstring)
Chat_TP2.py:1:0: C0103: Module name "Chat_TP2" doesn't conform to snake_case naming style (invalid-name)
Chat_TP2.py:13:0: C0116: Missing function or method docstring (missing-function-docstring)
Chat_TP2.py:16:4: W0602: Using global for 'conversation_history' but no assignment is done (global-variable-not-assigned)
Chat_TP2.py:52:0: C0116: Missing function or method docstring (missing-function-docstring)
Chat_TP2.py:53:4: W0602: Using global for 'conversation_history' but no assignment is done (global-variable-not-assigned)
Chat_TP2.py:72:15: W0718: Catching too general exception Exception (broad-exception-caught)
Chat_TP2.py:76:0: C0116: Missing function or method docstring (missing-function-docstring)
Chat_TP2.py:79:4: W0602: Using global for 'conversation_history' but no assignment is done (global-variable-not-assigned)

-----------------------------------
Your code has been rated at 7.73/10
```

b.

- Reduccion del tamaño de línea 85

```
    # This line parses the command-line arguments provided by the user using the parser
```

- Añadir docstring al modulo y funciones



```
chat_tp2.py > ...
"""
This Python program allows interaction with the chat GPT conversational engine from OpenAI.
The program accepts user queries, invokes the chatGPT API with
the provided query, and displays the obtained response on the screen.
It also provides additional functionalities such as error handling,
 the ability to retrieve and edit the last query,
and the option to enable a conversation mode.
Furthermore, measures are included to improve the code quality using pylint as a static analyzer.

Instructions:
1. Accepts user queries and sends them to chatGPT.
2. Handles errors using Try/Except structures.
3. Allows retrieval and editing of the last query using the "cursor Up" key.
4. Optionally enables a conversation mode.
5. Performs code quality metrics and proposes improvements.
6. Uses pylint to analyze the code and make necessary corrections.
"""
"""
Main function to handle command-line arguments and start the conversation mode.

This function parses the command-line arguments
to determine whether the conversation mode should be activated.
If the "--convers" flag is provided, conversation
mode is activated, otherwise, it's not activated.
It then calls the handle_user_interaction function accordingly.

Args:
    None

Returns:
    None
"""
# Global variable declaration to access the conversation history
```

- Renombrar el archivo de Chat_TP2, a chat_tp2 para seguir con el “snake_case naming style”
- Especificar las excepciones

```
# Define more specific exceptions to catch
SPECIFIC_EXCEPTIONS = (ConnectionError, TimeoutError)
```

```
except KeyboardInterrupt:
    print("\nSaliendo del programa.")
    break
except SPECIFIC_EXCEPTIONS as e:
    print("Error:", e)
```

- Eliminar las declaraciones globales de la variable “conversation_history” (declararla solo 1 vez)

```
--------------------------------------------------------------------
Your code has been rated at 10.00/10 (previous run: 10.00/10, +0.00)
```

3) Solicite a chatGPT que sugiera modificaciones y mejoras al programa Python elaborado.
   Recomendaciones de chatgpt:

- Tomar en cuenta errores de tipo: “ValueError” y “TypeError”

```python
# Define more specific exceptions to catch
SPECIFIC_EXCEPTIONS = (ConnectionError, TimeoutError, ValueError, TypeError)
```

- Simplificar funciones

```python
if conversation == 1:
    conversation_history.append(user_query)
    conversation_history.append(response.choices[0].message.content)

if conversation:
    conversation_history.append(user_query)
    conversation_history.append(response.choices[0].message.content)
```

```python
if conversation == 1:
    context = " ".join(conversation_history)  # Use conversation history as context
else:
    context = " "

context = " ".join(conversation_history) if conversation else ""
```

- Agregar manejo de excepciones a función chat_with_gpt

```
    try:
        # Generate response from the GPT-3 model
        response = client.chat.completions.create(
            model="gpt-3.5-turbo-0125",
            messages=[
                {"role": "system", "content": context},  # System context/message
                {"role": "user", "content": usertask},  # User task (if applicable)
                {"role": "user", "content": user_query},  # User query
            ],
            temperature=1,
            max_tokens=150,
            top_p=1,
            frequency_penalty=0,
            presence_penalty=0,
        )

        # Append the user query and GPT response to conversation history
        # Conditional working, or not, on conversation mode
        if conversation:
            conversation_history.append(user_query)
            conversation_history.append(response.choices[0].message.content)

        # Return the response from the GPT-3 model
        return response.choices[0].message.content
    except SPECIFIC_EXCEPTIONS as e:
        print("Error durante la conversación con GPT:", e)
        return None
```

Ejecución final de multimetric:

{

"comment_ratio": 56.214,

"cyclomatic_complexity": 7,

"fanout_external": 2,

"fanout_internal": 0,

"halstead_bugprop": 0.718,

"halstead_difficulty": 28.269,

"halstead_effort": 60857.963,

"halstead_timerequired": 3380.998,

"halstead_volume": 2152.799,

"lang": [

"Python"

```
    ],
    "loc": 82,
    "maintainability_index": 58.094,
    "operands_sum": 175,
    "operands_uniq": 65,
    "operators_sum": 160,
    "operators_uniq": 21,
    "pylint": 100.0,
    "tiobe": 87.727,
    "tiobe_compiler": 100.0,
    "tiobe_complexity": 18.182,
    "tiobe_coverage": 100.0,
    "tiobe_duplication": 100.0,
    "tiobe_fanout": 100.0,
    "tiobe_functional": 100.0,
    "tiobe_security": 100.0,
    "tiobe_standard": 100.0
  }
},
"overall": {
  "comment_ratio": 56.214,
  "cyclomatic_complexity": 7,
  "fanout_external": 2,
  "fanout_internal": 0,
  "halstead_bugprop": 0.718,
  "halstead_difficulty": 28.269,
  "halstead_effort": 60857.963,
  "halstead_timerequired": 3380.998,
  "halstead_volume": 2152.799,
  "loc": 82,
```

```
    "maintainability_index": 58.094,
    "operands_sum": 175,
    "operands_uniq": 65,
    "operators_sum": 160,
    "operators_uniq": 21,
    "pylint": 100.0,
    "tiobe": 87.727,
    "tiobe_compiler": 100.0,
    "tiobe_complexity": 18.182,
    "tiobe_coverage": 100.0,
    "tiobe_duplication": 100.0,
    "tiobe_fanout": 100.0,
    "tiobe_functional": 100.0,
    "tiobe_security": 100.0,
    "tiobe_standard": 100.0
  },
  "stats": {
    "max": {
      "comment_ratio": 56.214,
      "cyclomatic_complexity": 7,
      "fanout_external": 2,
      "fanout_internal": 0,
      "halstead_bugprop": 0.718,
      "halstead_difficulty": 28.269,
      "halstead_effort": 60857.963,
      "halstead_timerequired": 3380.998,
      "halstead_volume": 2152.799,
      "loc": 82,
      "maintainability_index": 58.094,
      "operands_sum": 175,
```

```
  "operands_uniq": 65,
  "operators_sum": 160,
  "operators_uniq": 21,
  "pylint": 100.0,
  "tiobe": 87.727,
  "tiobe_compiler": 100.0,
  "tiobe_complexity": 18.182,
  "tiobe_coverage": 100.0,
  "tiobe_duplication": 100.0,
  "tiobe_fanout": 100.0,
  "tiobe_functional": 100.0,
  "tiobe_security": 100.0,
  "tiobe_standard": 100.0
},
"mean": {
  "comment_ratio": 56.214,
  "cyclomatic_complexity": 7,
  "fanout_external": 2,
  "fanout_internal": 0,
  "halstead_bugprop": 0.718,
  "halstead_difficulty": 28.269,
  "halstead_effort": 60857.963,
  "halstead_timerequired": 3380.998,
  "halstead_volume": 2152.799,
  "loc": 82,
  "maintainability_index": 58.094,
  "operands_sum": 175,
  "operands_uniq": 65,
  "operators_sum": 160,
  "operators_uniq": 21,
```

```
  "pylint": 100.0,
  "tiobe": 87.727,
  "tiobe_compiler": 100.0,
  "tiobe_complexity": 18.182,
  "tiobe_coverage": 100.0,
  "tiobe_duplication": 100.0,
  "tiobe_fanout": 100.0,
  "tiobe_functional": 100.0,
  "tiobe_security": 100.0,
  "tiobe_standard": 100.0
},
"median": {
  "comment_ratio": 56.214,
  "cyclomatic_complexity": 7,
  "fanout_external": 2,
  "fanout_internal": 0,
  "halstead_bugprop": 0.718,
  "halstead_difficulty": 28.269,
  "halstead_effort": 60857.963,
  "halstead_timerequired": 3380.998,
  "halstead_volume": 2152.799,
  "loc": 82,
  "maintainability_index": 58.094,
  "operands_sum": 175,
  "operands_uniq": 65,
  "operators_sum": 160,
  "operators_uniq": 21,
  "pylint": 100.0,
  "tiobe": 87.727,
  "tiobe_compiler": 100.0,
```

```
  "tiobe_complexity": 18.182,
  "tiobe_coverage": 100.0,
  "tiobe_duplication": 100.0,
  "tiobe_fanout": 100.0,
  "tiobe_functional": 100.0,
  "tiobe_security": 100.0,
  "tiobe_standard": 100.0
},
"min": {
  "comment_ratio": 56.214,
  "cyclomatic_complexity": 7,
  "fanout_external": 2,
  "fanout_internal": 0,
  "halstead_bugprop": 0.718,
  "halstead_difficulty": 28.269,
  "halstead_effort": 60857.963,
  "halstead_timerequired": 3380.998,
  "halstead_volume": 2152.799,
  "loc": 82,
  "maintainability_index": 58.094,
  "operands_sum": 175,
  "operands_uniq": 65,
  "operators_sum": 160,
  "operators_uniq": 21,
  "pylint": 100.0,
  "tiobe": 87.727,
  "tiobe_compiler": 100.0,
  "tiobe_complexity": 18.182,
  "tiobe_coverage": 100.0,
  "tiobe_duplication": 100.0,
```

```
  "tiobe_fanout": 100.0,
  "tiobe_functional": 100.0,
  "tiobe_security": 100.0,
  "tiobe_standard": 100.0
}
```