25 de Mayo 385-Concepcion del Uruguay, CP: 3260. Dpto. Uruguay-
fcyt_cdelu@uader.edu.ar mailto:zubiaur79@yahoo.com.ar Tel-
03442-431442

TRABAJO PRACTICO

INGENIERIA EN SOFTWARE 2

UADER – Facultad de Ciencia y Tecnología

Profesor: Pedro Colla

Carrera: Licenciatura en Sistemas de Información

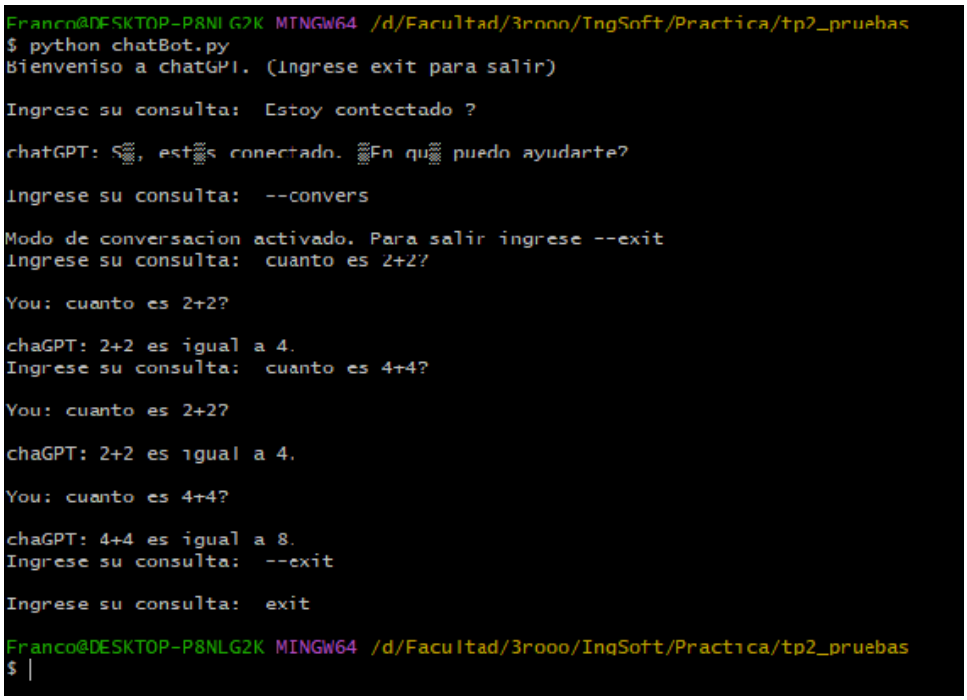
Alumno: Franco Leonardo Camen.

Ciclo lectivo: 2024.

PRUEBA DE EJECUCIÓN DEL CÓDIGO

Las consignas 1,2,3 y 4 están realizadas en el código. Anexo link a GitHub para ver el código en src/chatBot.py

https://github.com/FrancoCamen/UADER_IG2_-Camen-.git

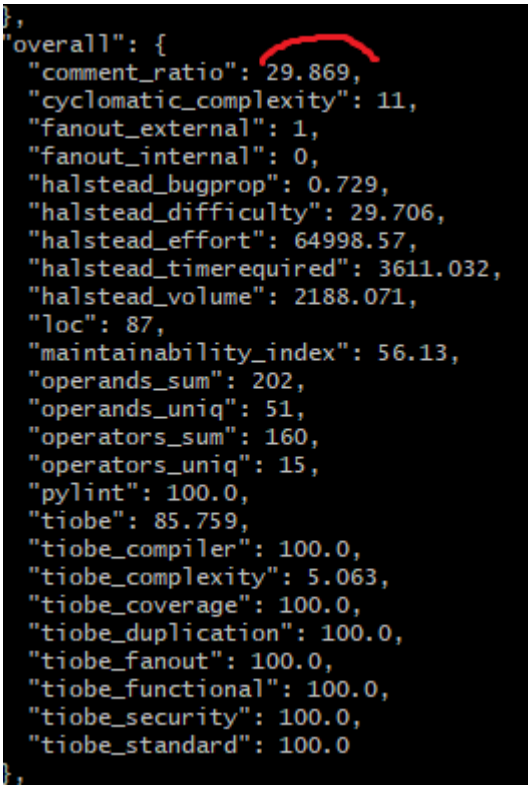


CONSIGNA 5

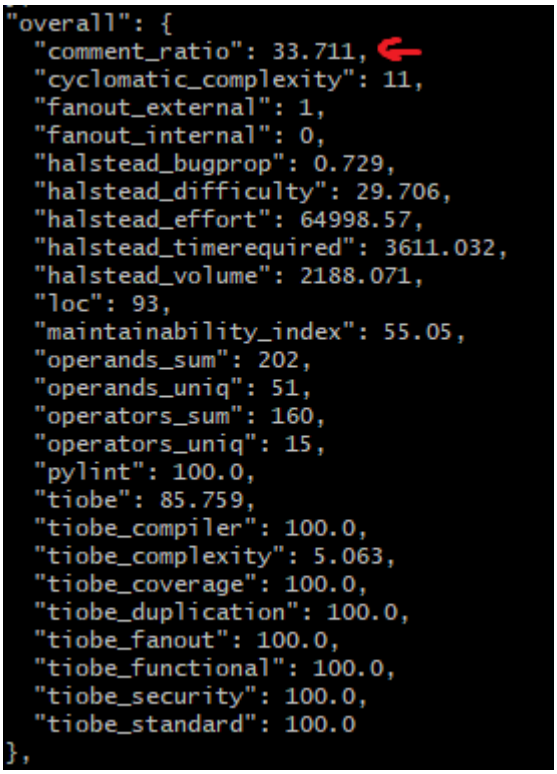
Ejecute el programa multimetric sobre la última versión que disponga del programa utilizado en éste practico.

PUNTO 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.



```
},
"overall": {
  "comment_ratio": 29.869,
  "cyclomatic_complexity": 11,
  "fanout_external": 1,
  "fanout_internal": 0,
  "halstead_bugprop": 0.729,
  "halstead_difficulty": 29.706,
  "halstead_effort": 64998.57,
  "halstead_timerequired": 3611.032,
  "halstead_volume": 2188.071,
  "loc": 87,
  "maintainability_index": 56.13,
  "operands_sum": 202,
  "operands_uniq": 51,
  "operators_sum": 160,
  "operators_uniq": 15,
  "pylint": 100.0,
  "tiobe": 85.759,
  "tiobe_compiler": 100.0,
  "tiobe_complexity": 5.063,
  "tiobe_coverage": 100.0,
  "tiobe_duplication": 100.0,
  "tiobe_fanout": 100.0,
  "tiobe_functional": 100.0,
  "tiobe_security": 100.0,
  "tiobe_standard": 100.0
},
```

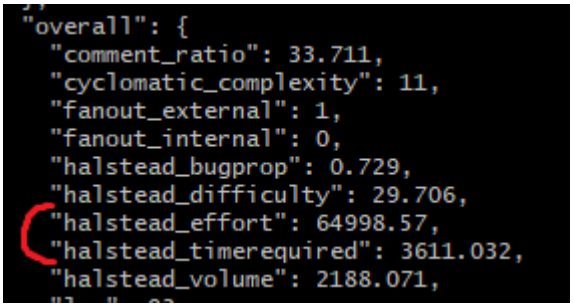
```
"overall": {
  "comment_ratio": 33.711,
  "cyclomatic_complexity": 11,
  "fanout_external": 1,
  "fanout_internal": 0,
  "halstead_bugprop": 0.729,
  "halstead_difficulty": 29.706,
  "halstead_effort": 64998.57,
  "halstead_timerequired": 3611.032,
  "halstead_volume": 2188.071,
  "loc": 93,
  "maintainability_index": 55.05,
  "operands_sum": 202,
  "operands_uniq": 51,
  "operators_sum": 160,
  "operators_uniq": 15,
  "pylint": 100.0,
  "tiobe": 85.759,
  "tiobe_compiler": 100.0,
  "tiobe_complexity": 5.063,
  "tiobe_coverage": 100.0,
  "tiobe_duplication": 100.0,
  "tiobe_fanout": 100.0,
  "tiobe_functional": 100.0,
  "tiobe_security": 100.0,
  "tiobe_standard": 100.0
},
```


Después de agregar algunos comentarios al código explicando mejor como funciona este, se aumento el parametro comment_ratio.

PUNTO B

Explore el significado de y luego compare los valores de halstead_effort y halstead_timerequired con los que efectivamente le tomó el programa.

- **Halstead_effort:** Representa la cantidad de esfuerzo teórico necesario para escribir el programa. Cuanto mayor sea el valor de Halstead Effort, mayor será la complejidad del programa en términos de la cantidad de operadores y operandos utilizados. Esta métrica proporciona una idea del esfuerzo cognitivo requerido para comprender y mantener el código.
- **Halstead_timerequired:** Estima el tiempo requerido para escribir el programa en función del esfuerzo de Halstead. Cuanto mayor sea el valor de Halstead Time Required, mayor será el tiempo necesario para escribir el programa.



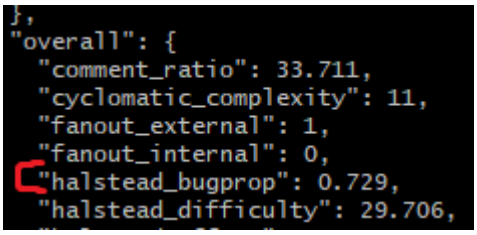
- **Halstead_effort:** 64998.57
- **Halstead_timerequired:** 3611.032

El valor dado por halstead required da 3.600 segundos aprox que serian 60minutos, estableciendo que se demoraría 1 hora es escribir el código, en mi caso esto no es asi ya que me llevo mas tiempo, esto puede reflejar que la demora para escribir el código es relativa según la experiencia del programador.

PUNTO 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).

- Halstead_bugprod: Proporciona una estimación de la proporción de errores esperados en el código en relación con su complejidad, según la teoría de Halstead. Una proporción más alta indica una mayor probabilidad de errores en el código debido a su complejidad.



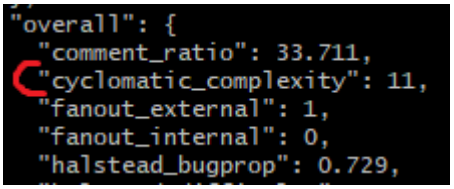
0.729

Comparando el valor halstead_bugprop con la cantidad de errores que tuve luego de ejecutar por primera vez estos valores son relativamente bajos, dado que no tuve demasiados errores luego de hacer correr por primera vez el programa, la mayor razón de errores fueron antes de lograr correr debido a error de sintaxis y planteamiento del código.

PUNTO D

Que estrategias cree que se pueden aplicar para reducir el índice de McCabe (cyclomatic_complexity) en un 10%.

- Cyclomatic_complexity: Cuantifica la cantidad de decisiones lógicas en un programa. Cuantas más decisiones lógicas haya en el código (como bucles, condicionales, casos switch, etc.), mayor será la complejidad ciclomática.



11

Estrategias para reducir la complejidad ciclomatica:

- Reducir la complejidad de cada función individual dividiéndola en funciones más pequeñas y específicas.
- Reducir cantidad de estructuras de control anidadas (bucles o if anidados) para evitar profundidad excesiva de anidamiento.
- Evitar el uso de bucles for y while con condiciones complejas.
- Refactorizar código redundante.

Mi código es particular tiene un bucle while muy profundo que aumenta la complejidad, para solucionar esto se puede desglosar las estructuras condicionales que contiene y encapsularla en una función específica.

CONSIGNA 6

Instale un analizador estático de código denominado pylint (mediante el comando pip install pylint) y realice las siguientes acciones.

PUNTO A

Ejecute el programa pylint sobre el programa python desarrollado como parte de las consignas del punto -4- anterior.

```
chatBot.py:24:20: C0303: Trailing whitespace (trailing-whitespace)
chatBot.py:30:50: C0303: Trailing whitespace (trailing-whitespace)
chatBot.py:34:0: C0303: Trailing whitespace (trailing-whitespace)
chatBot.py:56:0: C0301: Line too long (107/100) (line-too-long)
chatBot.py:57:43: C0303: Trailing whitespace (trailing-whitespace)
chatBot.py:63:0: C0303: Trailing whitespace (trailing-whitespace)
chatBot.py:64:0: C0301: Line too long (107/100) (line-too-long)
chatBot.py:78:0: C0301: Line too long (107/100) (line-too-long)
chatBot.py:88:0: C0301: Line too long (121/100) (line-too-long)
chatBot.py:90:0: C0301: Line too long (102/100) (line-too-long)
chatBot.py:101:0: C0303: Trailing whitespace (trailing-whitespace)
chatBot.py:110:0: C0305: Trailing newlines (trailing-newlines)
chatBot.py:1:0: C0114: Missing module docstring (missing-module-docstring)
chatBot.py:1:0: C0103: Module name "chatBot" doesn't conform to snake_case namin
g style (invalid-name)
chatBot.py:11:0: C0103: Constant name "api_key" doesn't conform to UPPER_CASE na
ming style (invalid-name)
chatBot.py:16:0: C0116: Missing function or method docstring (missing-function-d
ocstring)
chatBot.py:16:13: W0621: Redefining name 'consulta' from outer scope (line 43) (
redefined-outer-name)
chatBot.py:19:8: W0621: Redefining name 'response' from outer scope (line 16) (r
edefined-outer-name)
chatBot.py:31:11: W0718: Catching too general exception Exception (broad-excepti
on-caught)
chatBot.py:16:0: R1710: Either all return statements in a function should return
 an expression, or none of them should. (inconsistent-return-statements)
chatBot.py:39:0: C0103: Constant name "condicion" doesn't conform to UPPER_CASE
naming style (invalid-name)
chatBot.py:40:6: C0121: Comparison 'condicion == True' should be 'condicion is T
rue' if checking for the singleton value True, or 'condicion' if testing for tru
thiness (singleton-comparison)
chatBot.py:49:8: R1723: Unnecessary "else" after "break", remove the "else" and
de-indent the code inside it (no-else-break)
chatBot.py:64:20: R1723: Unnecessary "else" after "break", remove the "else" and
 de-indent the code inside it (no-else-break)
chatBot.py:65:24: W0104: Statement seems to have no effect (pointless-statement)
chatBot.py:65:24: C0121: Comparison 'condicion == False' should be 'condicion is
 False' if checking for the singleton value False, or 'not condicion' if testing
 for falsiness (singleton-comparison)
chatBot.py:86:24: W0702: No exception type(s) specified (bare-except)
chatBot.py:98:16: W0702: No exception type(s) specified (bare-except)

-----------------------------------
Your code has been rated at 4.40/10
```

Luego de revisar los comentarios de pylint se solucionar los posibles errores del código.

```
Franco@DESKTOP-P8NLG2K MINGW64 /d/Facultad/3rooo/IngSoft/Practica/tp2_pruebas
$ pylint chat_bot.py

--------------------------------------------------------------------
Your code has been rated at 10.00/10 (previous run: 9.81/10, +0.19)
```

CONSIGNA 7

Solicite a chatGPT que sugiera modificaciones y mejoras al programa Python elaborado.

Las mejoras sugeridas por chatGPT son las siguientes:

- **Comentarios claros y concisos:** Los comentarios deben ser claros y concisos. Puedes agregar comentarios breves para explicar el propósito de secciones específicas de código o el funcionamiento de funciones importantes.
- **Nombre de variables y constantes:** Usa nombres de variables y constantes más descriptivos y en minúsculas. Esto hace que el código sea más legible y fácil de entender.
- **Manejo de excepciones:** Especifica el tipo de excepción que deseas capturar en lugar de capturar Exception genérica. Esto hace que el manejo de errores sea más específico y reduce la posibilidad de ocultar problemas importantes.
- **Organización del código:** Organiza el código en bloques lógicos y utiliza espacios en blanco y sangrías adecuadas para mejorar la legibilidad.
- **Eliminación de código muerto:** Elimina comentarios y código muerto que no aportan valor al programa.