

ActiviTY 4.2   
PROGRAMMING EXERCISES 1

TC4017 Software testing and quality assurance

|  |  |
| --- | --- |
|  | Student: |
| Arturo Alfonso Gallardo Jasso | A01795510 |
|  |  |
|  | Professors: |

Dr. Gerardo Padilla Zárate

Sajith Delgado Miranda

# **Activity instructions**

1. Implement the programs from this document using the Python language.
2. Follow the PEP-8 coding standard.
3. Verify the correct execution of the programs by generating tests of each exercise using the indicated resources. Document the results.
4. Install the pylint package using PIP, <https://pypi.org/project/pylint/>
5. Verify that your programs do not generate errors or issues using pylint.

There are 5 kind of message types :

\* (C) convention, for programming standard violation

\* (R) refactor, for bad code smell

\* (W) warning, for python specific problems

\* (E) error, for probable bugs in the code

\* (F) fatal, if an error occurred which prevented pylint from doing

1. Fix all the issues found by pylint and verify that your program continues to work correctly.
2. When finished, upload your programs to your personal repository.
3. The project should be named: StudentID\_ActivityNumberA4.2
4. Submit the repository link in the Canvas assignment.
5. Also, upload the source files of the task to Canvas.

# **Repository**

The project folder and a copy of this PDF report can be found in the following link:

https://github.com/AAGJ89/MNA\_TC4017\_SW.git

# **Platform details**

Python 3.10.11

Visual Studio Code 1.96.4

# **Problem 1.** **Compute statistics**

**1.1 Software Requirements**

Req1. The program shall be invoked from a command line. The program shall receive a file as parameter. The file will contain a list of items (presumable numbers).

Req 2. The program shall compute all descriptive statistics from a file containing numbers. The results shall be print on a screen and on a file named *StatisticsResults.txt*. **All computation MUST be calculated using the basic algorithms, not functions or libraries.**

The descriptive statistics are mean, median, mode, standard deviation, and variance.

Req 3. The program shall include the mechanism to handle invalid data in the file. Errors should be displayed in the console and the execution must continue.

Req 4. The name of the program shall be computeStatistics.py

Req 5. The minimum format to invoke the program shall be as follows:

python computeStatistics.py fileWithData.txt

Req 6. The program shall manage files having from hundreds of items to thousands of items.

Req 7. The program should include at the end of the execution the time elapsed for the execution and calculus of the data. This number shall be included in the results file and on the screen.

Req 8. Be compliant with PEP8.

**1.2 Code Highlights**

* The “computeStatistics” program successfully invokes the file using the required format.
* The program computes all descriptive statistics using basic algorithms, prints the results on screen and stores them into “StatisticsResults.txt” file.
* The program prints the time elapsed, from the beginning of the execution until complete the storage of the data.
* Refer to Figure 1 to see compliance of the program with Requirements 1, 2, 4, 5, 7.

A screen shot of a computer

Description automatically generated

Figure 1

* The program stores the results in “StatisticsResults.txt”, see Figure 2.

A screenshot of a computer

Description automatically generated

Figure 2

* The program successfully manages different error types complying with Requirement 3:
  + Error 1: Wrong format to invoke the program. See Figure 3.
  + Error 2: File not found. See Figure 4.
  + Error 3: No data in the file. File “TC8.txt” was added to folder. See Figure 5.
  + Error 4: Invalid data. See Figure 6.



Figure 3



Figure 4



Figure 5

A screen shot of a computer

Description automatically generated

Figure 6

* Program can invoke any of the Files in the folder complying with Requirement 6.

**1.3 Fixing the source data to match results**

The following changes were made based on the “A4.2.P1.Results-errata” file:

* TC1. The reference file has 400 counts, therefore, 405 is fixed by deleting the typo “s”.

|  |  |
| --- | --- |
| **TC1** descriptive statistics **before** fix | **TC1** descriptive statistics **after** fix |
| Mean: 241.91  Median: 239.00  Mode: 393  Variance: 21086.31  Standard Deviation: 145.21  Time elapsed: 0.0010 seconds | Mean: 242.32  Median: 239.50  Mode: 393  Variance: 21099.92  Standard Deviation: 145.26  Time elapsed: 0.0061 seconds |

Table 1

* TC2. No changes.

|  |  |
| --- | --- |
| **TC2** descriptive statistics |  |
| Mean: 250.78  Median: 247.00  Mode: 230  Variance: 20785.37  Standard Deviation: 144.17  Time elapsed: 0.0021 seconds |  |

Table 2

* TC3. No changes.

|  |  |
| --- | --- |
| **TC3** descriptive statistics |  |
| Mean: 249.78  Median: 249.00  Mode: 94  Variance: 21117.28  Standard Deviation: 145.32  Time elapsed: 0.0115 seconds |  |

Table 3

* TC4. No changes.

|  |
| --- |
| **TC4** descriptive statistics |
| Mean: 149.00  Median: 147.75  Mode: 123.75  Variance: 17007.92  Standard Deviation: 130.41  Time elapsed: 0.0061 seconds |

Table 4

* TC5. Invalid data: ABA, 23,45, 11;54, ll. Using the file as reference, it shows that the values from TC5, they are the same than TC1, these are the changes: ABA = 410; 23,45 = 275; 11,54 = 356, ll = 148. Even with the changes, the Mean and Median are very different.

|  |  |
| --- | --- |
| **TC5** descriptive statistics **before** fix | **TC5** descriptive statistics **after** fix |
| Mean: 241.50  Median: 241.00  Mode: 466.0  Variance: 21160.02  Standard Deviation: 145.46  Time elapsed: 0.0000 seconds | Mean: 242.21  Median: 246.00  Mode: 466.0  Variance: 21052.53  Standard Deviation: 145.09  Time elapsed: 0.0079 seconds |

Table 5

* TC6. No changes.

|  |  |
| --- | --- |
| **TC6** descriptive statistics |  |
| Mean: 187906599279774728192.00  Median: 188008049965542998016.00  Mode: 3.74846462174395e+20  Variance: 11530904699530646862954721780958962384896.00  Standard Deviation: 107382050173809999872.00  Time elapsed: 0.0173 seconds |  |

Table 6

* TC7. Invalid data: ABBA, ERROR. This values were changed to 0.

|  |  |
| --- | --- |
| **TC7** descriptive statistics **before** fix | **TC7** descriptive statistics **after** fix |
| Mean: 247467395499714904064.00  Median: 246640973074290016256.00  Mode: 4.99994082497151e+20  Variance: 20910793147136488598117820783212011454464.00  Standard Deviation:  144605647009847050240.00  Time elapsed: 0.0149 seconds | Mean: 247428634845709139968.00  Median: 246611679700462993408.00  Mode: 0.0  Variance: 20917108399182968491730835396544742555648.00  Standard Deviation:  144627481479776059392.00  Time elapsed: 0.0158 seconds |

Table 7

**1.4 Pylint report**

The report of the first version of the code is given by Figure 7.

A computer screen shot of a black and white screen

Description automatically generated

Figure 7

The score was 8.40 of 10.00. The next section describes the changes to comply with PEP8.

(C) Convention. Style-related issues:

* C0303 - Trailing whitespace. Extra space removed at the end of line 65.
* C0304 - Missing final newline. Added line to row 87, after the change, another C0304 opportunity appeared in row 96 and a new line was added after row 96.
* C0114 – Missing module docstring. I added a quick description of the code and owner at the top using the triple quotes.
* C0301 - Line too long and C0209 – Formatting a regular string which could be an f-string. For these two, the results instruction from line 75 was modified to make it short by lines, less than 100 by recommendations of PEP8 and used f-string by recommendation of the tool.
* C0103: - Invalid name. The name of the file "computeStatistics" is part of the requirement, therefore, a condition was added at the beginning of the code to skip the compliance with the snake\_case naming style.
* C0103: Constant name "item\_qty" and “mid”number” do not conform to UPPER\_CASE naming style. The change was made.

The new score after the changes was 9.80 as described above, the new missing final newline was resolved getting a 10.00 as score.

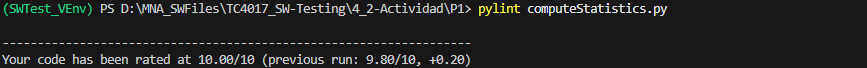


Figure 8

The program meets Requirement 8.

# **Problem 2. Converter**

**2.1 Software Requirements**

Req1. The program shall be invoked from a command line. The program shall receive a file as parameter. The file will contain a list of items (presumable numbers).

Req 2. The program shall convert the numbers to binary and hexadecimal base. The results shall be print on a screen and on a file named *ConvertionResults.txt*. **All computation MUST be calculated using the basic algorithms, not functions or libraries.**

Req 3. The program shall include the mechanism to handle invalid data in the file. Errors should be displayed in the console and the execution must continue.

Req 4. The name of the program shall be convertNumbers.py

Req 5. The minimum format to invoke the program shall be as follows: python convertNumbers.py fileWithData.txt

Req 6. The program shall manage files having from hundreds of items to thousands of items.

Req 7. The program should include at the end of the execution the time elapsed for the execution and calculus of the data. This number shall be included in the results file and on the screen.

Req 8. Be compliant with PEP8.

**2.2 Code Highlights**

* The “convertNumbers” program successfully invokes the file using the required format.
* The program converts the source data into binary and hexadecimal, prints the results on screen and stores them into “ConvertionResults.txt” file.
* The program prints the time elapsed, from the beginning of the execution until complete the storage of the data.
* Refer to Figure 9 and Figure 10 to see compliance of the program with Requirements 1, 2, 4, 5, 7.

A screenshot of a computer

Description automatically generated

Figure 9

A black background with white text

Description automatically generatedFigure 10

* The program stores the results in “StatisticsResults.txt”, see Figure 11.

A screenshot of a computer

Description automatically generated

Figure 11

* The program successfully manages different error types complying with Requirement 3:
  + Error 1: Wrong format to invoke the program. See Figure 12.
  + Error 2: File not found. See Figure 4.



Figure 12



Figure 13

* Program can invoke any of the Files in the folder complying with Requirement 6.

**2.3 Results**

The results from the conversion are observed in Figure 9, 14, 15 and 16. Being the negative numbers a challenge and match the values from the reference file “A4.2.P2.Results”.

TC2:

A screenshot of a computer

Description automatically generated

Figure 14

TC3:

A screen shot of a computer

Description automatically generated

Figure 15

TC4 was able to continue, even with three errors:

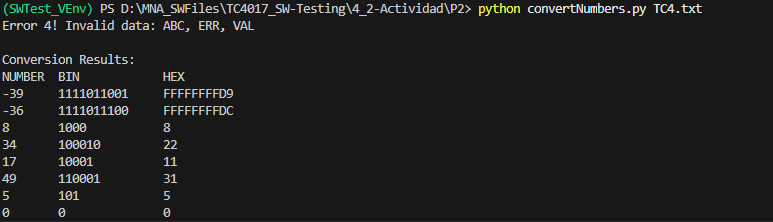


Figure 16

**2.4 Pylint report**

The report of the first version of the code is given by Figure 17.

A screen shot of a computer program

Description automatically generated

Figure 17

The score is 8.64 of 10.00. Higher than Problem 1. The next section describes the changes to comply with PEP8.

(W) Warning, for python specific problems:

* W0311 - Bad indentation. Correct space was given for if, else routine. The same mistake occurred for line 52 and 62.

(C) Convention. Style-related issues:

* C0303 - Trailing whitespace. Extra space removed at the end of line 62 and 64. The same for row 95.
* C0301 – Line too long. Lines 78 and 85 were eliminated, instead a f-string commands were implemented.
* C0305 – Trailing newlines. Line added at the end.

The new score is 10.00 after all previous changes.

A screen shot of a black background

Description automatically generated

Figure 18

The program meets Requirement 8.

# **Problem 3. Count Words**

**3.1 Software Requirements**

Req1. The program shall be invoked from a command line. The program shall receive a file as parameter. The file will contain a list of items (presumable numbers).

Req 2. The program shall identify all distinct words and the frequency of them (how many times the word “X” appears in the file). The results shall be print on a screen and on a file named *WordCountResults.txt.* **All computation MUST be calculated using the basic algorithms, not functions or libraries.**

Req 3. The program shall include the mechanism to handle invalid data in the file. Errors should be displayed in the console and the execution must continue.

Req 4. The name of the program shall be wordCount.py

Req 5. The minimum format to invoke the program shall be as follows:

python wordCount.py fileWithData.txt

Req 6. The program shall manage files having from hundreds of items to thousands of items.

Req 7. The program should include at the end of the execution the time elapsed for the execution and calculus of the data. This number shall be included in the results file and on the screen.

Req 8. Be compliant with PEP8.

**3.2 Code Highlights**

* The “wordCount” program successfully invokes the file using the required format.
* The program counts the words to know the frequency and print in descending order using basic algorithms, and stores the values into “WordCountResults.txt” file.
* The program prints the time elapsed, from the beginning of the execution until complete the storage of the data.
* Refer to Figure 19 and Figure 20 to see compliance of the program with Requirements 1, 2, 4, 5, 7.

A screen shot of a computer

Description automatically generated

Figure 19

A screen shot of a computer

Description automatically generated

Figure 20

* The program stores the results in “StatisticsResults.txt”, see Figure 21.

A screenshot of a computer

Description automatically generated

Figure 21

* The program successfully manages different error types complying with Requirement 3:
  + Error 1: Wrong format to invoke the program. See Figure 22.
  + Error 2: File not found. See Figure 23.
  + We do not have cases of Invalid data from this exercise.



Figure 22



Figure 23

* Program can invoke any of the Files in the folder complying with Requirement 6.

**3.3 Results**

The results from the counter are observed in Figure 24, 25, 19, 20, 26, 27, 28, 29, 30 and 31.

TC1:

A computer screen with white text

Description automatically generated

Figure 24

A screen shot of a computer program

Description automatically generated

Figure 25

TC3:

A screen shot of a computer

Description automatically generated

Figure 26

A screenshot of a computer

Description automatically generated

Figure 27

TC4:

A screen shot of a computer

Description automatically generated

Figure 28

A screenshot of a computer program

Description automatically generated

Figure 29

A white background with black and white clouds

Description automatically generated with medium confidence

Figure 30

A screen shot of a computer

Description automatically generated

Figure 31

**3.4 Pylint report**

The report of the first version of the code is given by Figure 32.

A screenshot of a computer program

Description automatically generated

Figure 32

The score is 9.63 of 10.00. Higher than Problem 2. The next section describes the changes to comply with PEP8.

(W) Warning, for python specific problems:

* W0311 - Bad indentation. Correct space was given, then, a new finding appeared related to exceeding the max length of the line, but it was fixed.

(C) Convention. Style-related issues:

C0410 – Multiple imports. A mistake at the beginning of the code was fixed.

The new score is 10.00 after all previous changes.

A black screen with white text

Description automatically generated

Figure 33

The program meets Requirement 8.

# **Conclusions**

In these three exercises, three exercises were resolved enhancing the understanding on code development based on requirements. Some of these requirements are not clear, or need more description of what is expected. Also, there are reference files that caused more confusion. I made the mistake of not going back to the requester to clarify these points. Even with this opportunity, I strongly believe the three programs meet the expectations and fall perfectly into a first learning cycle with the requester. Iterative and Incremental Development Model in software development. Then, in new cycles, target a more robust solution that fits the purpose desired.

During the development and testing of the three exercises, it followed the PEP8 guidelines, which promote writing clean, readable, and maintainable code. This included using Pylint to get proper indentation, variable names, consistent spacing, and clear comments to improve code clarity. VSCode/Python already provides warnings and exposes errors when there is a command, or syntaxis, or missing letter in the code during the development, it also shows errors if trying to execute it, this helps a lot prior applying Pylint to reduce non-complaint wording or structure. However, by reviewing the PEP8 documentation, we can reflect on how the standard helps maintain code quality in larger projects.

# **References**

Spolsky, J. (2005). *Making Wrong Code Look Wrong – Joel on Software*.

Van Rossum, G., Warsaw, B. & Coghlan, Alyssa (2013). PEP 8 – Style Guide for Python Cod. Convención de codificación de Python - PEP8 <https://peps.python.org/pep-0008/>

Lutz, M. (2013). *Learning Python, 5th Edition.* 5ed.

Pylint Tutorial – How to Write Clean Python