



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
6. Fix all the issues found by pylint and verify that your program continues to work correctly.
7. When finished, upload your programs to your personal repository.
8. The project should be named: StudentID_ActivityNumberA4.2
9. Submit the repository link in the Canvas assignment.
10. 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
```

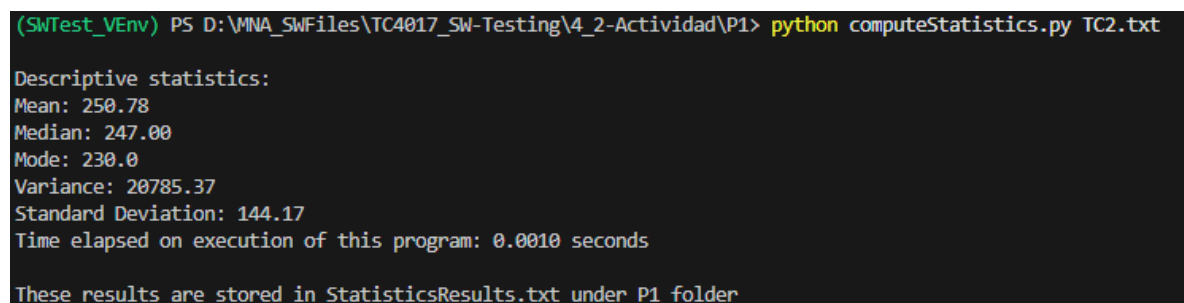
Req 6. The program shall manage files having from hundreds of items to thousands of items.

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.



```
(SwTest_VEnv) PS D:\MNA_SWFiles\TC4017_SW-Testing\4_2-Actividad\P1> python computeStatistics.py TC2.txt

Descriptive statistics:
Mean: 250.78
Median: 247.00
Mode: 230.0
Variance: 20785.37
Standard Deviation: 144.17
Time elapsed on execution of this program: 0.0010 seconds

These results are stored in StatisticsResults.txt under P1 folder
```

Figure 1

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

Name	Date modified	Type	Size
.DS_Store	1/30/2025 9:36 PM	DS_STORE File	7 KB
.gitignore	1/31/2025 1:03 AM	Git Ignore Source ...	1 KB
A4.2.P1.Results-errata	2/1/2025 10:20 PM	Text Document	1 KB
computeStatistics	2/1/2025 10:06 PM	Python Source File	3 KB
StatisticsResults	2/1/2025 10:16 PM	Text Document	1 KB
TC1	2/1/2025 10:41 PM	Text Document	2 KB

Figure 2

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

```
(SwTest_VEnv) PS D:\MNA_SWFiles\TC4017_SW-Testing\4_2-Actividad\P1> python computeStatistics.py
Error 1! Format to invoke the program is: python computeStatistics.py TCX.txt
```

Figure 3

```
(SwTest_VEnv) PS D:\MNA_SWFiles\TC4017_SW-Testing\4_2-Actividad\P1> python computeStatistics.py TC.txt
Error 2! File not found: TC.txt
```

Figure 4

```
(SwTest_VEnv) PS D:\MNA_SWFiles\TC4017_SW-Testing\4_2-Actividad\P1> python computeStatistics.py TC8.txt
Error 3! No data in the file
```

Figure 5

```
(SwTest_VEnv) PS D:\MNA_SWFiles\TC4017_SW-Testing\4_2-Actividad\P1> python computeStatistics.py TC5.txt
Error 4! Invalid data: ABA, 23,45, 11;54, 11

Descriptive statistics:
Mean: 241.50
Median: 241.00
Mode: 466.0
Variance: 21160.02
Standard Deviation: 145.46
Time elapsed: 0.0040 seconds

These results are stored in StatisticsResults.txt under P1 folder
```

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: 209107931471364885981178207832120 11454464.00 Standard Deviation: 144605647009847050240.00 Time elapsed: 0.0149 seconds	Mean: 247428634845709139968.00 Median: 246611679700462993408.00 Mode: 0.0 Variance: 209171083991829684917308353965447 42555648.00 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.

```
(SwTest_VEnv) PS D:\MNA_SWFiles\TC4017_SW-Testing\4_2-Actividad\VP1> pylint computeStatistics.py
***** Module computeStatistics
computeStatistics.py:65:16: C0303: Trailing whitespace (trailing-whitespace)
computeStatistics.py:75:0: C0301: Line too long (146/100) (line-too-long)
computeStatistics.py:87:0: C0304: Final newline missing (missing-final-newline)
computeStatistics.py:1:0: C0114: Missing module docstring (missing-module-docstring)
computeStatistics.py:1:0: C0103: Module name "computeStatistics" doesn't conform to snake_case naming style (invalid-name)
computeStatistics.py:45:0: C0103: Constant name "item_qty" doesn't conform to UPPER_CASE naming style (invalid-name)
computeStatistics.py:52:0: C0103: Constant name "mid number" doesn't conform to UPPER_CASE naming style (invalid-name)
computeStatistics.py:75:10: C0209: Formatting a regular string which could be an f-string (consider-using-f-string)

-----
Your code has been rated at 8.40/10 (previous run: 8.40/10, +0.00)
```

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.

```
(SwTest_VEnv) PS D:\MNA_SWFiles\TC4017_SW-Testing\4_2-Actividad\VP1> pylint computeStatistics.py
-----
Your code has been rated at 10.00/10 (previous run: 9.80/10, +0.20)
```

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 *ConversionResults.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 “ConversionResults.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.

```
(SWTest_VEnv) PS D:\MNA_SWFiles\TC4017_SW-Testing\4_2-Actividad\P2> python convertNumbers.py TC1.txt

Conversion Results:
NUMBER      BIN                                HEX
6980368     11010101000001100010000          6A8310
5517055     10101000010111011111111          542EFF
1336159     101000110001101011111          14635F
6750185     11001101111111111101001          66FFE9
1771937     110110000100110100001          1B09A1
360952      1011000000111111000          581F8
5672561     10101101000111001110001          568E71
916583      11011111110001100111          DFC67
```

Figure 9

```
2250854      1000100101100001100110          225866
Time elapsed on execution of this program: 0.0115 seconds

These results are stored in ConversionResults.txt under P2 folder
```

Figure 10

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








Name	Date modified	Type	Size
 A4.2.P2.Results	1/30/2025 10:15 PM	RESULTS File	17 KB
 ConversionResults	2/1/2025 11:31 PM	Text Document	11 KB
 convertNumbers	2/1/2025 9:13 PM	Python Source File	3 KB
 TC1	1/30/2025 10:15 PM	Text Document	2 KB
 TC2	1/30/2025 10:15 PM	Text Document	2 KB
 TC3	1/30/2025 10:15 PM	Text Document	1 KB
 TC4	1/30/2025 10:15 PM	Text Document	1 KB

Figure 11

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

```
(SwTest_VEnv) PS D:\MNA_SWFiles\TC4017_SW-Testing\4_2-Actividad\P2> python convertNumbers.py
Error 1! Format to invoke the program is: python computeStatistics.py TCX.txt
```

Figure 12

```
(SwTest_VEnv) PS D:\MNA_SWFiles\TC4017_SW-Testing\4_2-Actividad\P2> python convertNumbers.py TC.txt
Error 2! File not found: TC.txt
```

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:

```
(SwTest_VEnv) PS D:\MNA_SWFiles\TC4017_SW-Testing\4_2-Actividad\P2> python convertNumbers.py TC2.txt
```

Conversion Results:

NUMBER	BIN	HEX
7116776	1101100100101111101000	6C97E8
1666340	110010110110100100100	196D24
8886983	100001111001101011000111	879AC7
839365	11001100111011000101	CCEC5
924280	11100001101001111000	E1A78
1026310	11111010100100000110	FA906
1615293	110001010010110111101	18A5BD
1063875	100000011101111000011	103BC3

Figure 14

TC3:

```
(SwTest_VEnv) PS D:\MNA_SWFiles\TC4017_SW-Testing\4_2-Actividad\P2> python convertNumbers.py TC3.txt
```

Conversion Results:

NUMBER	BIN	HEX
-39	1111011001	FFFFFFFD9
-36	1111011100	FFFFFFFDC
8	1000	8
34	100010	22
17	10001	11
49	110001	31
5	101	5
39	100111	27

Figure 15

TC4 was able to continue, even with three errors:

```
(SwTest_VEnv) PS D:\MNA_SWFiles\TC4017_SW-Testing\4_2-Actividad\P2> python convertNumbers.py TC4.txt
```

Error 4! Invalid data: ABC, ERR, VAL

Conversion Results:

NUMBER	BIN	HEX
-39	1111011001	FFFFFFFD9
-36	1111011100	FFFFFFFDC
8	1000	8
34	100010	22
17	10001	11
49	110001	31
5	101	5
0	0	0

Figure 16

2.4 Pylint report

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

```
(SwTest_VEnv) PS D:\MNA_SWFiles\TC4017_SW-Testing\4_2-Actividad\P2> pylint convertNumbers.py
***** Module convertNumbers
convertNumbers.py:52:0: W0311: Bad indentation. Found 5 spaces, expected 8 (bad-indentation)
convertNumbers.py:62:53: C0303: Trailing whitespace (trailing-whitespace)
convertNumbers.py:62:0: W0311: Bad indentation. Found 5 spaces, expected 8 (bad-indentation)
convertNumbers.py:64:66: C0303: Trailing whitespace (trailing-whitespace)
convertNumbers.py:78:0: C0301: Line too long (119/100) (line-too-long)
convertNumbers.py:85:0: C0301: Line too long (125/100) (line-too-long)
convertNumbers.py:95:0: C0303: Trailing whitespace (trailing-whitespace)
convertNumbers.py:99:0: C0305: Trailing newlines (trailing-newlines)

-----
Your code has been rated at 8.64/10 (previous run: 8.64/10, +0.00)
```

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.

```
(SwTest_VEnv) PS D:\MNA_SWFiles\TC4017_SW-Testing\4_2-Actividad\P2> pylint convertNumbers.py

-----
Your code has been rated at 10.00/10 (previous run: 9.83/10, +0.17)
```

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
```

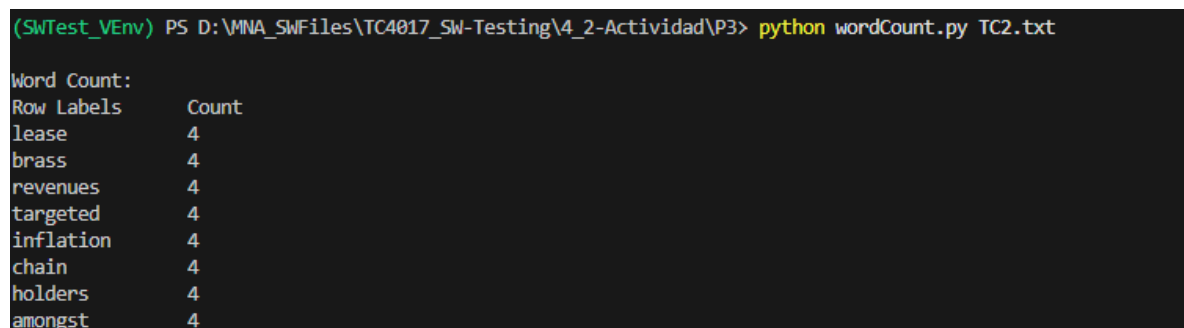
Req 6. The program shall manage files having from hundreds of items to thousands of items.

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.



```
(SWTest_VEnv) PS D:\MNA_SWFiles\TC4017_SW-Testing\4_2-Actividad\P3> python wordCount.py TC2.txt
```

Word Count:	
Row Labels	Count
lease	4
brass	4
revenues	4
targeted	4
inflation	4
chain	4
holders	4
amongst	4

Figure 19

```

initiative      1
variety         1
ion             1
interface       1
remained        1
icons           1
excessive       1
jon             1
Time elapsed on execution of this program: 0.0089 seconds

These results are stored in WordCountResults.txt under P3 folder

```

Figure 20

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




Name	Date modified	Type	Size
 WordCountResults	2/2/2025 10:27 AM	Text Document	3 KB
 wordCount	2/2/2025 10:25 AM	Python Source File	3 KB
 TC5	1/30/2025 10:14 PM	Text Document	37 KB

Figure 21

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

```

(SwTest_VEnv) PS D:\MNA_SWFiles\TC4017_SW-Testing\4_2-Actividad\P3> python wordCount.py
Error 1! Format to invoke the program is: python computeStatistics.py TCX.txt

```

Figure 22

```

(SwTest_VEnv) PS D:\MNA_SWFiles\TC4017_SW-Testing\4_2-Actividad\P3> python wordCount.py TC.txt
Error 2! File not found: TC.txt

```

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:

```
(SwTest_VEnv) PS D:\MNA_SWFiles\TC4017_SW-Testing\4_2-Actividad\P3> python wordCount.py TC1.txt

Word Count:
Row Labels    Count
conservative  2
mother        1
tions         1
pin           1
sure          1
regulatory    1
shower        1
uni           1
```

Figure 24

```
lies          1
seats         1
worse         1
guestbook     1
influences    1
kodak         1
significance   1
coastal       1
Time elapsed on execution of this program: 0.0050 seconds

These results are stored in WordCountResults.txt under P3 folder
```

Figure 25

TC3:

```
(SwTest_VEnv) PS D:\MNA_SWFiles\TC4017_SW-Testing\4_2-Actividad\P3> python wordCount.py TC3.txt

Word Count:
Row Labels    Count
notice        3
flood         2
pottery       2
charity        2
suggestion     2
pairs          2
blues          2
pipe           2
```

Figure 26

```
census        1
uc             1
declined       1
index          1
equipped       1
src            1
gradually      1
placing        1
Time elapsed on execution of this program: 0.0343 seconds

These results are stored in WordCountResults.txt under P3 folder
```

Figure 27

TC4:

```
(SwTest_VEnv) PS D:\MNA_SWFiles\TC4017_SW-Testing\4_2-Actividad\P3> python wordCount.py TC4.txt

Word Count:
Row Labels      Count
started         3
literally       2
ringtone        2
za              2
reached         2
crazy           2
javascript       2
annual          2
```

Figure 28

```
calling         1
mailman         1
extent          1
corpus          1
sv              1
wishlist        1
expired         1
circular        1
Time elapsed on execution of this program: 0.0846 seconds

These results are stored in WordCountResults.txt under P3 folder
```

Figure 29

Row Labels	Count
wilderness	5
managed	5
schools	5
pets	5
kg	5
ggs	4
keeping	4
travelling	4

Figure 30

```
seo            1
pounds         1
suggesting     1
texas          1
postposted    1
realty         1
vaccine        1
relocation     1
Time elapsed on execution of this program: 0.8274 seconds

These results are stored in WordCountResults.txt under P3 folder
```

Figure 31

3.4 Pylint report

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

```
(SwTest_VEnv) PS D:\MNA_SWFiles\TC4017_SW-Testing\4_2-Actividad\P3> pylint wordCount.py
***** Module wordCount
wordCount.py:61:0: W0311: Bad indentation. Found 9 spaces, expected 12 (bad-indentation)
wordCount.py:10:0: C0410: Multiple imports on one line (sys, time) (multiple-imports)

-----
Your code has been rated at 9.63/10 (previous run: 9.63/10, +0.00)
```

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.

```
(SwTest_VEnv) PS D:\MNA_SWFiles\TC4017_SW-Testing\4_2-Actividad\P3> pylint wordCount.py

-----
Your code has been rated at 10.00/10 (previous run: 9.82/10, +0.18)
```

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 syntax, 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