

México, 28 de enero de 2024

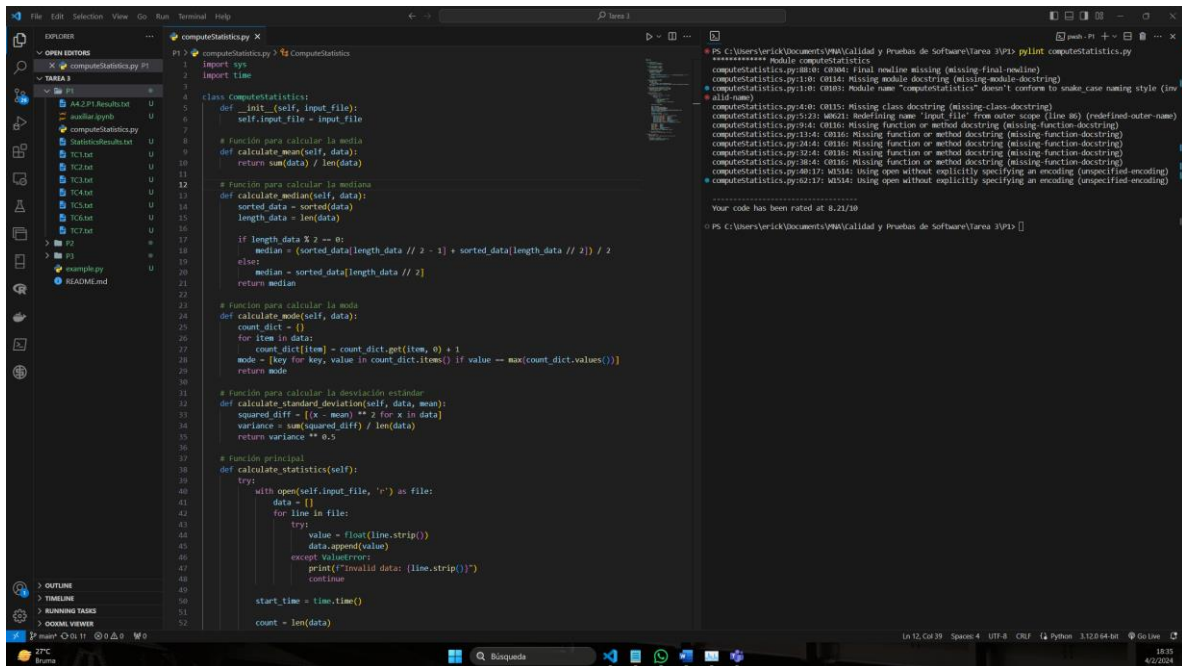
## Actividad 4.2 | Ejercicio de programación 1

GitHub: [https://github.com/ErickHCerecedo/A01066428\\_A4.2](https://github.com/ErickHCerecedo/A01066428_A4.2)

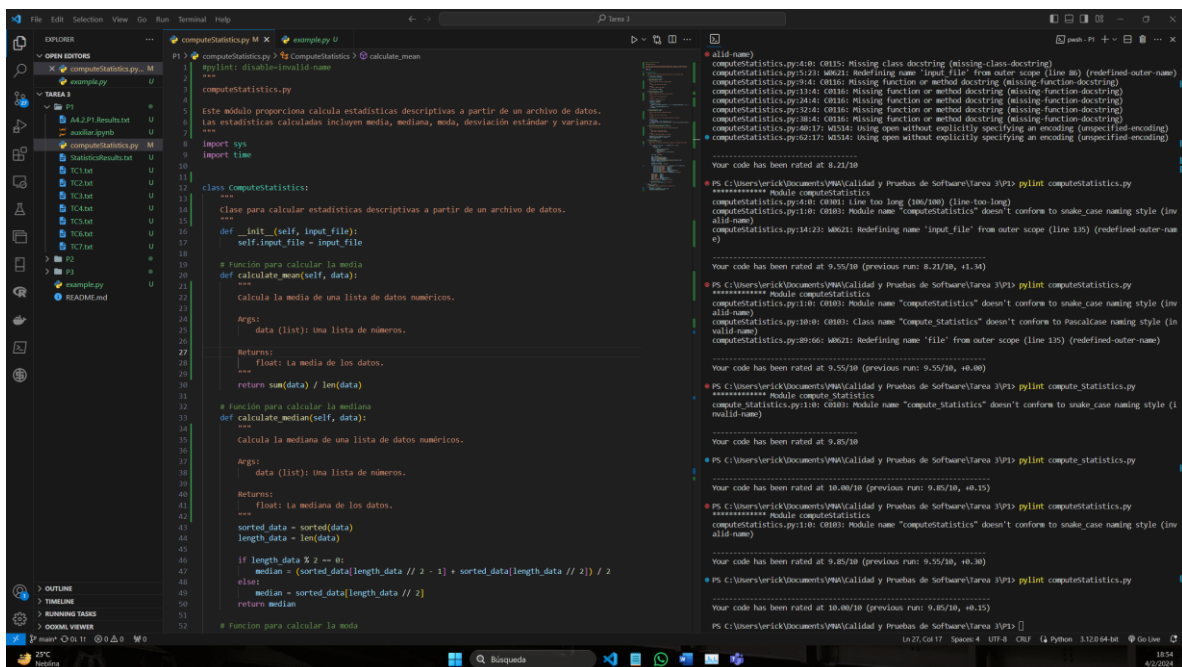
En la siguiente actividad empleando Python se da solución a 3 problemas de programación siguiendo las pautas del estándar de programación PEP-8, una de las herramientas empleadas para el análisis de redacción y estilo es PyLint para asegurar que el programa no genere errores o problemas, en este reporte se encuentra documentado el proceso de creación y corrección de cada uno de los problemas.

Problem 1: Compute Statistics	
Description	<p>Requirement 1. 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).</p> <p>Requirement 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. <b>All computation MUST be calculated using the basic algorithms, not functions or libraries.</b></p> <p>The descriptive statistics are mean, median, mode, standard deviation, and variance.</p> <p>Requirement 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.</p> <p>Requirement 4. The name of the program shall be computeStatistics.py</p> <p>Requirement 5. The minimum format to invoke the program shall be as follows: python computeStatistics.py fileWithData.txt</p> <p>Requirement 6. The program shall manage files having from hundreds of items to thousands of items.</p> <p>Requirement 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.</p> <p>Requirement 8. Be compliant with PEP8.</p>

## 1. Implementación del problema 1 y primera revisión usando PyLint:

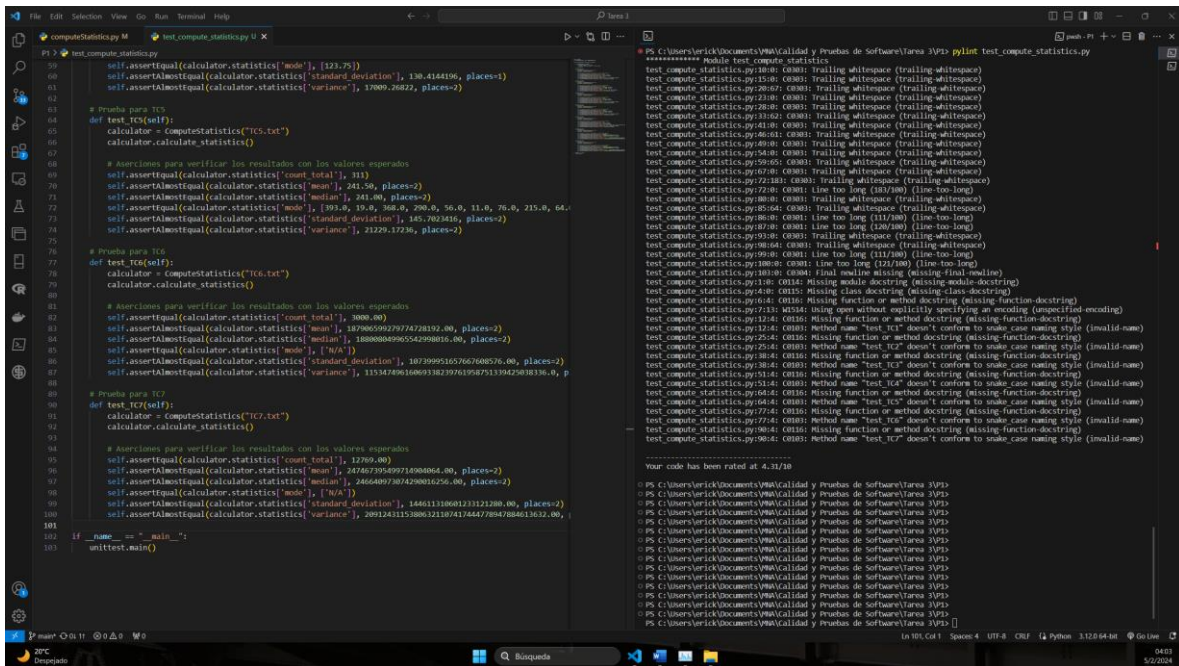


## 2. Corrección del código usando PyLint:

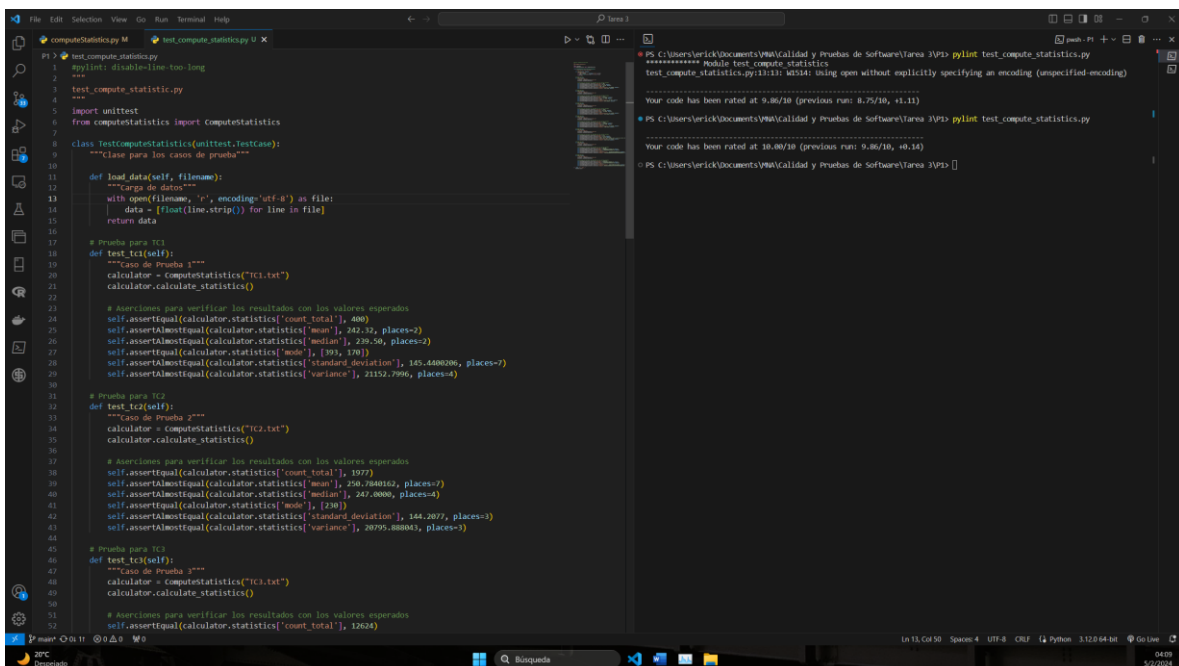


**Nota \*** El nombre del archivo `computeStatistics.py` no cumple con el estándar `snake_case` para nombrar archivos, pero debido a ser un requerimiento se usó el comando “disable” de PyLint para ignorar esta advertencia.

### 3. Creación un archivo de pruebas unitarias:



### 4. Corrección de código usando PyLint:



**Nota\*** algunos de los valores proporcionados en el archivo de resultados contenían errores, ya que algunos conjuntos de datos eran multimodales (Contienen más de un valor en moda), los resultados contenidos en casos de pruebas fueron comprobados en otra herramienta de análisis de datos.

## 5. Casos de prueba aprobados:

The screenshot shows a PyCharm IDE with a Python test file named `test_compute_statistics.py` on the left and its output on the right. The test file contains several test cases for a `compute_statistics` module. The output on the right shows the results of these tests, including statistics like count, mean, median, mode, standard deviation, and variance for different datasets. The tests are all passing, as indicated by the green checkmarks and the 'OK' status at the bottom.

```
1 # test_compute_statistics.py
2 """
3 test_compute_statistics.py
4 """
5 import unittest
6 from compute_statistics import ComputeStatistics
7
8 class TestComputeStatistics(unittest.TestCase):
9     """Clase para los casos de prueba"""
10
11     def load_data(self, filename):
12         """Carga de datos"""
13         with open(filename, 'r', encoding='utf-8') as file:
14             data = [float(line.strip()) for line in file]
15         return data
16
17     # Prueba para TC1
18     def test_tc1(self):
19         """Caso de Prueba 1"""
20         calculator = ComputeStatistics("TC1.txt")
21         calculator.calculate_statistics()
22
23         # Aserciones para verificar los resultados con los valores esperados
24         self.assertEqual(calculator.statistics['count_total'], 400)
25         self.assertAlmostEqual(calculator.statistics['mean'], 242.32, places=2)
26         self.assertAlmostEqual(calculator.statistics['median'], 249.50, places=2)
27         self.assertEqual(calculator.statistics['mode'], [193, 170])
28         self.assertAlmostEqual(calculator.statistics['standard deviation'], 145.4408206, places=7)
29         self.assertAlmostEqual(calculator.statistics['variance'], 21152.7996, places=4)
30
31     # Prueba para TC2
32     def test_tc2(self):
33         """Caso de Prueba 2"""
34         calculator = ComputeStatistics("TC2.txt")
35         calculator.calculate_statistics()
36
37         # Aserciones para verificar los resultados con los valores esperados
38         self.assertEqual(calculator.statistics['count_total'], 1977)
39         self.assertAlmostEqual(calculator.statistics['mean'], 250.780162, places=7)
40         self.assertAlmostEqual(calculator.statistics['median'], 247.0000, places=4)
41         self.assertEqual(calculator.statistics['mode'], [230])
42         self.assertAlmostEqual(calculator.statistics['standard deviation'], 144.2077, places=3)
43         self.assertAlmostEqual(calculator.statistics['variance'], 20795.88803, places=3)
44
45     # Prueba para TC3
46     def test_tc3(self):
47         """Caso de Prueba 3"""
48         calculator = ComputeStatistics("TC3.txt")
49         calculator.calculate_statistics()
50
51         # Aserciones para verificar los resultados con los valores esperados
52         self.assertEqual(calculator.statistics['count_total'], 12624)
```

Output:

```
.,COUNT: 1977
MEAN: 250.78
MEDIAN: 247.00
MODE: 230.0
SD: 144.21
VARIANCE: 20795.88
Elapsed Time: 0.004152089555053711 seconds

.,COUNT: 12624
MEAN: 249.78
MEDIAN: 249.00
MODE: 94.0
SD: 145.12
VARIANCE: 21118.95
Elapsed Time: 0.017102439934692383 seconds

.,COUNT: 12624
MEAN: 149.00
MEDIAN: 147.75
MODE: 123.75
SD: 126.42
VARIANCE: 17069.27
Elapsed Time: 0.010151254774365234 seconds

.,COUNT: 311
MEAN: 281.50
MEDIAN: 282.00
MODE: 301.0, 19.0, 368.0, 290.0, 56.0, 11.0, 76.0, 215.0, 64.0, 375.0, 466.0, 277.0, 211.0, 46.0, 278.0, 170.0, 166.0, 96.0, 208.0
SD: 147.70
VARIANCE: 21829.17
Elapsed Time: 0.000617805488957031 seconds

.,COUNT: 3000
MEAN: 187900599279774728132.00
MEDIAN: 180000000000000000000.00
MODE: N/A
SD: 2873999265767680876.00
VARIANCE: 1153474061608913823976195875133425038136.00
Elapsed Time: 0.006587382237915839 seconds

.,COUNT: 12769
MEAN: 287402795499734000004.00
MEDIAN: 28660973074200016256.00
MODE: N/A
SD: 14861118801121121200.00
VARIANCE: 209124111330652110342444778947884613632.00
Elapsed Time: 0.013000130056030273 seconds

Run 7 tests in 0.001s
OK
PS C:\Users\jerick\Documents\YMA\Calidad y Pruebas de Software\Tarea 3\P2>
```

## Problem 2: Converter

Requirement 1. 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).

Requirement 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.**

Requirement 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.

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

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

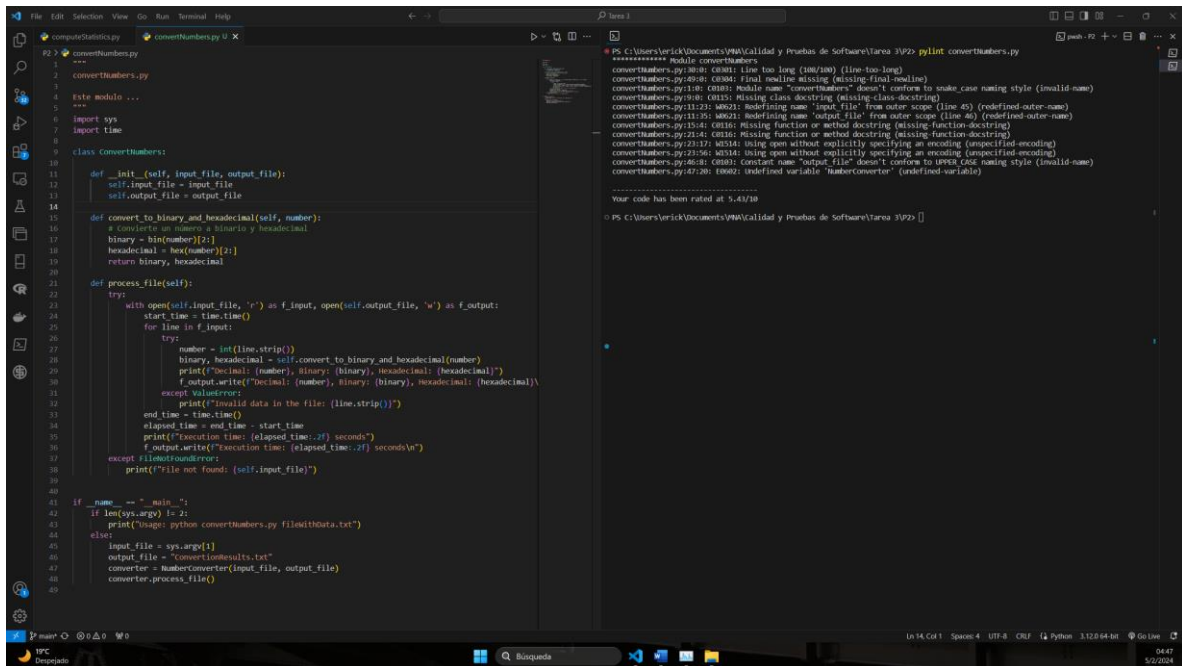
Requirement 6. The program shall manage files having from hundreds of items to thousands of items.

Requirement 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.

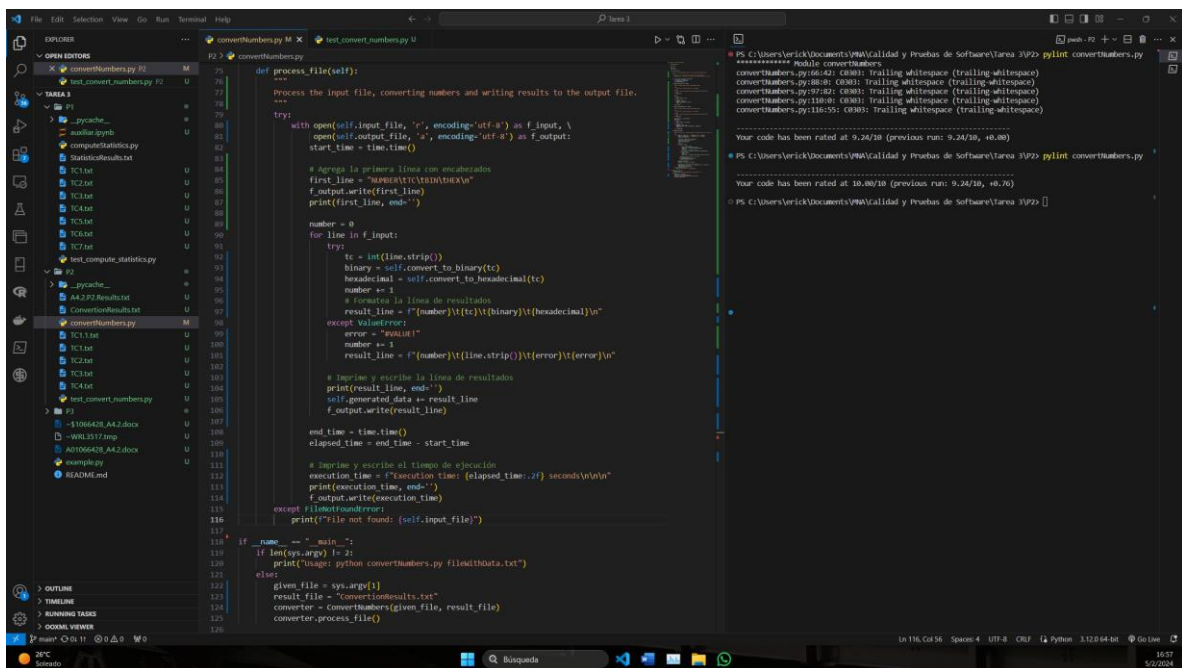
Requirement 8. Be compliant with PEP8.

Description

## 1. Implementación del problema 2 y primera revisión usando PyLint:

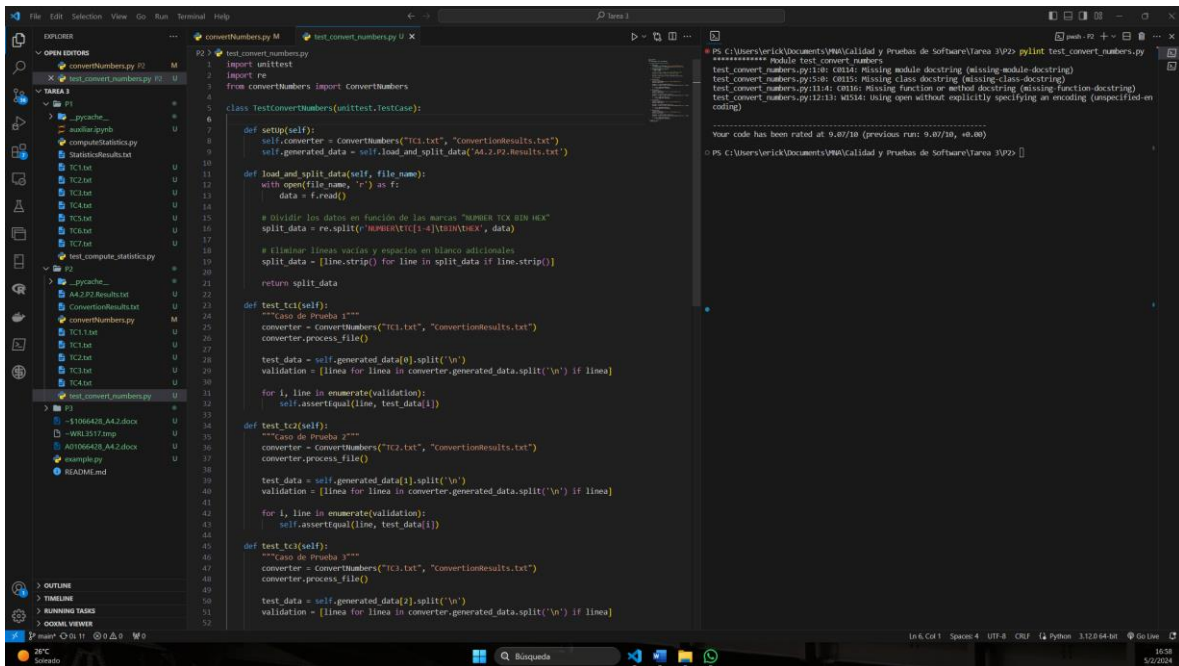


## 2. Corrección del código usando PyLint:





### 3. Creación un archivo de pruebas unitarias:



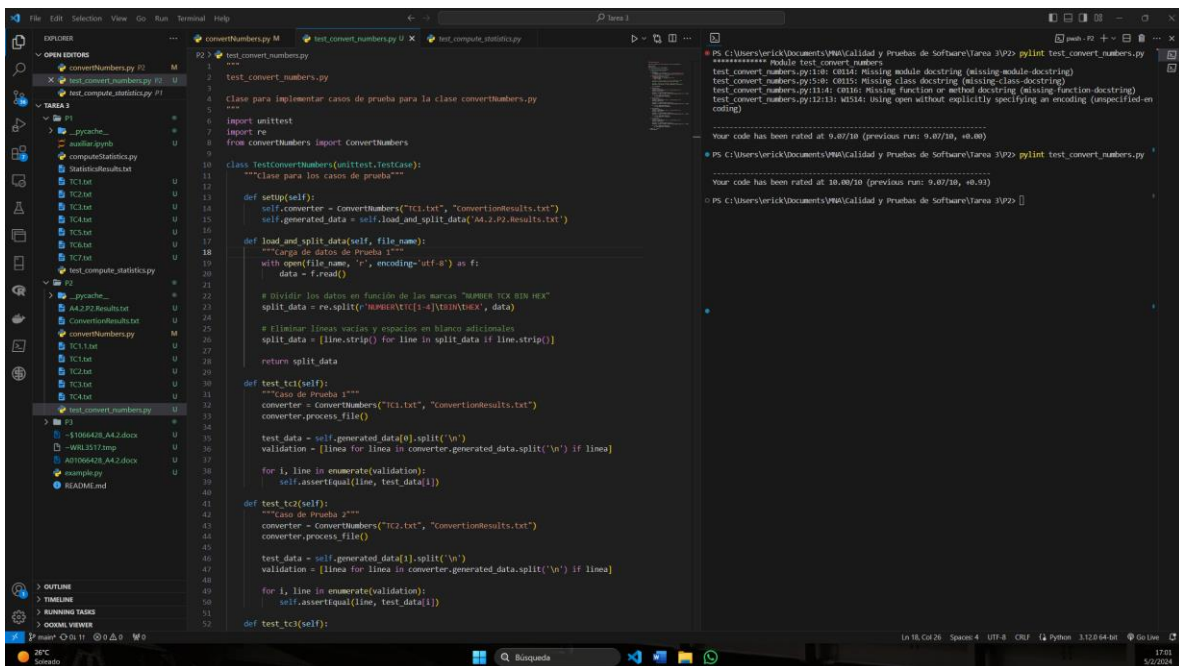
```
1 import unittest
2 import re
3 from convertNumbers import ConvertNumbers
4
5 class TestConvertNumbers(unittest.TestCase):
6
7     def setUp(self):
8         self.converter = ConvertNumbers("TC1.txt", "ConversionResults.txt")
9         self.generated_data = self.load_and_split_data("AA2.P2.Results.txt")
10
11     def load_and_split_data(self, file_name):
12         with open(file_name, 'r') as f:
13             data = f.read()
14
15         # Dividir los datos en función de las marcas "NUMBER TCC BIN HEX"
16         split_data = re.split(r"NUMBER TCC BIN HEX", data)
17
18         # Eliminar líneas vacías y espacios en blanco adicionales
19         split_data = [line.strip() for line in split_data if line.strip()]
20
21         return split_data
22
23     def test_tc1(self):
24         """Caso de Prueba 1"""
25         converter = ConvertNumbers("TC1.txt", "ConversionResults.txt")
26         converter.process_file()
27
28         test_data = self.generated_data[0].split('\n')
29         validation = [linea for linea in converter.generated_data.split('\n') if linea]
30
31         for i, line in enumerate(validation):
32             self.assertEqual(line, test_data[i])
33
34     def test_tc2(self):
35         """Caso de Prueba 2"""
36         converter = ConvertNumbers("TC2.txt", "ConversionResults.txt")
37         converter.process_file()
38
39         test_data = self.generated_data[1].split('\n')
40         validation = [linea for linea in converter.generated_data.split('\n') if linea]
41
42         for i, line in enumerate(validation):
43             self.assertEqual(line, test_data[i])
44
45     def test_tc3(self):
46         """Caso de Prueba 3"""
47         converter = ConvertNumbers("TC3.txt", "ConversionResults.txt")
48         converter.process_file()
49
50         test_data = self.generated_data[2].split('\n')
51         validation = [linea for linea in converter.generated_data.split('\n') if linea]
```

PyLint output:

```
PS C:\Users\verick\Documents\VMW\Calidad y Pruebas de Software\Tarea 3\2> pylint test_convert_numbers.py
***** Module test_convert_numbers
test_convert_numbers.py:18: CR14: Missing module docstring (missing-module-docstring)
test_convert_numbers.py:5:0: CR15: Missing class docstring (missing-class-docstring)
test_convert_numbers.py:11:4: CR16: Missing function or method docstring (missing-function-docstring)
test_convert_numbers.py:12:13: W554: using open without explicitly specifying an encoding (unspecified-encoding)

Your code has been rated at 9.00/10 (previous run: 9.00/10, +0.00)
```

### 4. Corrección de código usando PyLint:



```
1 import unittest
2 import re
3 from convertNumbers import ConvertNumbers
4
5 """Clase para los casos de prueba"""
6
7 class TestConvertNumbers(unittest.TestCase):
8
9     def setUp(self):
10         self.converter = ConvertNumbers("TC1.txt", "ConversionResults.txt")
11         self.generated_data = self.load_and_split_data("AA2.P2.Results.txt")
12
13     def load_and_split_data(self, file_name):
14         """Carga de datos de prueba"""
15         with open(file_name, 'r', encoding='utf-8') as f:
16             data = f.read()
17
18         # Dividir los datos en función de las marcas "NUMBER TCC BIN HEX"
19         split_data = re.split(r"NUMBER TCC BIN HEX", data)
20
21         # Eliminar líneas vacías y espacios en blanco adicionales
22         split_data = [line.strip() for line in split_data if line.strip()]
23
24         return split_data
25
26     def test_tc1(self):
27         """Caso de Prueba 1"""
28         converter = ConvertNumbers("TC1.txt", "ConversionResults.txt")
29         converter.process_file()
30
31         test_data = self.generated_data[0].split('\n')
32         validation = [linea for linea in converter.generated_data.split('\n') if linea]
33
34         for i, line in enumerate(validation):
35             self.assertEqual(line, test_data[i])
36
37     def test_tc2(self):
38         """Caso de Prueba 2"""
39         converter = ConvertNumbers("TC2.txt", "ConversionResults.txt")
40         converter.process_file()
41
42         test_data = self.generated_data[1].split('\n')
43         validation = [linea for linea in converter.generated_data.split('\n') if linea]
44
45         for i, line in enumerate(validation):
46             self.assertEqual(line, test_data[i])
47
48     def test_tc3(self):
49         """Caso de Prueba 3"""
50         converter = ConvertNumbers("TC3.txt", "ConversionResults.txt")
51         converter.process_file()
52
53         test_data = self.generated_data[2].split('\n')
54         validation = [linea for linea in converter.generated_data.split('\n') if linea]
```

PyLint output:

```
PS C:\Users\verick\Documents\VMW\Calidad y Pruebas de Software\Tarea 3\2> pylint test_convert_numbers.py
***** Module test_convert_numbers
test_convert_numbers.py:18: CR14: Missing module docstring (missing-module-docstring)
test_convert_numbers.py:5:0: CR15: Missing class docstring (missing-class-docstring)
test_convert_numbers.py:11:4: CR16: Missing function or method docstring (missing-function-docstring)
test_convert_numbers.py:12:13: W554: using open without explicitly specifying an encoding (unspecified-encoding)

Your code has been rated at 10.00/10 (previous run: 9.00/10, +0.93)
```

## 5. Casos de prueba aprobados:

```

1  # coding: utf-8
2  """
3  Clase para implementar casos de prueba para la clase ConvertNumbers.py
4  """
5  import unittest
6  import re
7  from convertNumbers import ConvertNumbers
8
9  class TestConvertNumbers(unittest.TestCase):
10     """Clase para los casos de prueba"""
11
12     def setUp(self):
13         self.converter = ConvertNumbers("TC1.txt", "ConversionResults.txt")
14         self.generated_data = self.load_and_split_data("Ad2.92.Results.txt")
15
16     def load_and_split_data(self, file_name):
17         """Carga de datos de prueba"""
18         with open(file_name, 'r', encoding='utf-8') as f:
19             data = f.read()
20
21         # Dividir los datos en función de las marcas "NUMBER TCX BIN HEX"
22         split_data = re.split("NUMBER TCX BIN HEX", data)
23
24         # Eliminar líneas vacías y espacios en blanco adicionales
25         split_data = [line.strip() for line in split_data if line.strip()]
26
27         return split_data
28
29     def test_tc1(self):
30         """Caso de prueba 1"""
31         converter = ConvertNumbers("TC1.txt", "ConversionResults.txt")
32         converter.process_file()
33
34         test_data = self.generated_data[0].split('\n')
35         validation = [line for line in converter.generated_data.split('\n') if line]
36
37         for i, line in enumerate(validation):
38             self.assertEqual(line, test_data[i])
39
40     def test_tc2(self):
41         """Caso de prueba 2"""
42         converter = ConvertNumbers("TC2.txt", "ConversionResults.txt")
43         converter.process_file()
44
45         test_data = self.generated_data[1].split('\n')
46         validation = [line for line in converter.generated_data.split('\n') if line]
47
48         for i, line in enumerate(validation):
49             self.assertEqual(line, test_data[i])
50
51     def test_tc3(self):
52

```

Execution time: 0.04 seconds

NUMBER	TC	BIN	HEX
1	-13	1111110011	FFFFFFF3
2	-35	1111100000	FFFFFFF0
3	8	1111111100	FFFFFFF8
4	34	1000010 22	FFFFFFF7
5	17	10001 11	FFFFFFF6
6	49	110001 11	FFFFFFF5
7	5	101	FFFFFFF4
8	40C	POWELL POWELL	FFFFFFF3
9	0	0	FFFFFFF2
10	0	0	FFFFFFF1
11	12	1100 C	FFFFFFF0
12	6	111111010	FFFFFFEF
13	27	11011 10	FFFFFFEE
14	-4	1111111100	FFFFFFED
15	-38	1110110100	FFFFFFEC
16	26	11010 1A	FFFFFFEB
17	40	110001 11	FFFFFFEA
18	29	11101 10	FFFFFFE9
19	42	101010 2A	FFFFFFE8
20	-16	1111100000	FFFFFFE7
21	100	POWELL POWELL	FFFFFFE6
22	14	100010 22	FFFFFFE5
23	20	10100 1A	FFFFFFE4
24	0	0	FFFFFFE3
25	25	11001 10	FFFFFFE2
26	4	10101 10	FFFFFFE1
27	1	1	FFFFFFE0
28	-46	1110100010	FFFFFFDF
29	-46	1110100010	FFFFFFDE
30	29	11101 10	FFFFFFDD
31	13	100001 21	FFFFFFDC
32	29	11101 10	FFFFFFDB
33	26	11010 1A	FFFFFFDA
34	-5	1111111011	FFFFFFD9
35	-36	1110111100	FFFFFFD8
36	12	1100 C	FFFFFFD7
37	45	101101 2D	FFFFFFD6
38	-28	1110011110	FFFFFFD5
39	0	0	FFFFFFD4
40	0	0	FFFFFFD3
41	-6	1111111010	FFFFFFD2
42	100	POWELL POWELL	FFFFFFD1

Execution time: 0.01 seconds

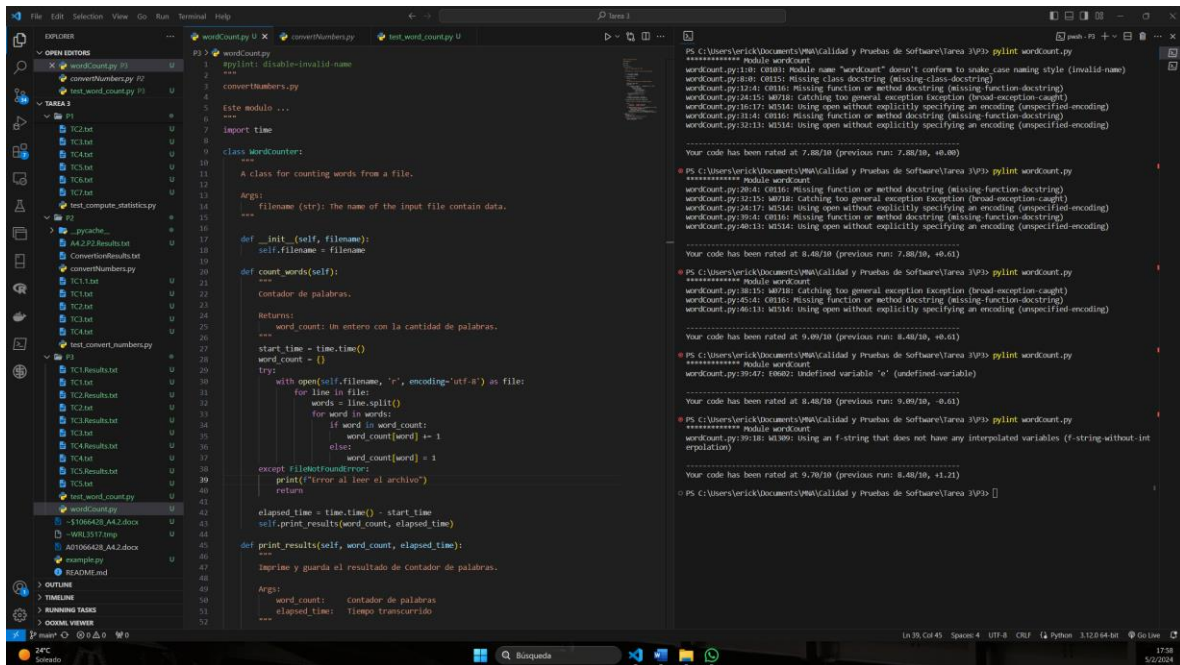
OK

Run 4 tests in 0.137s

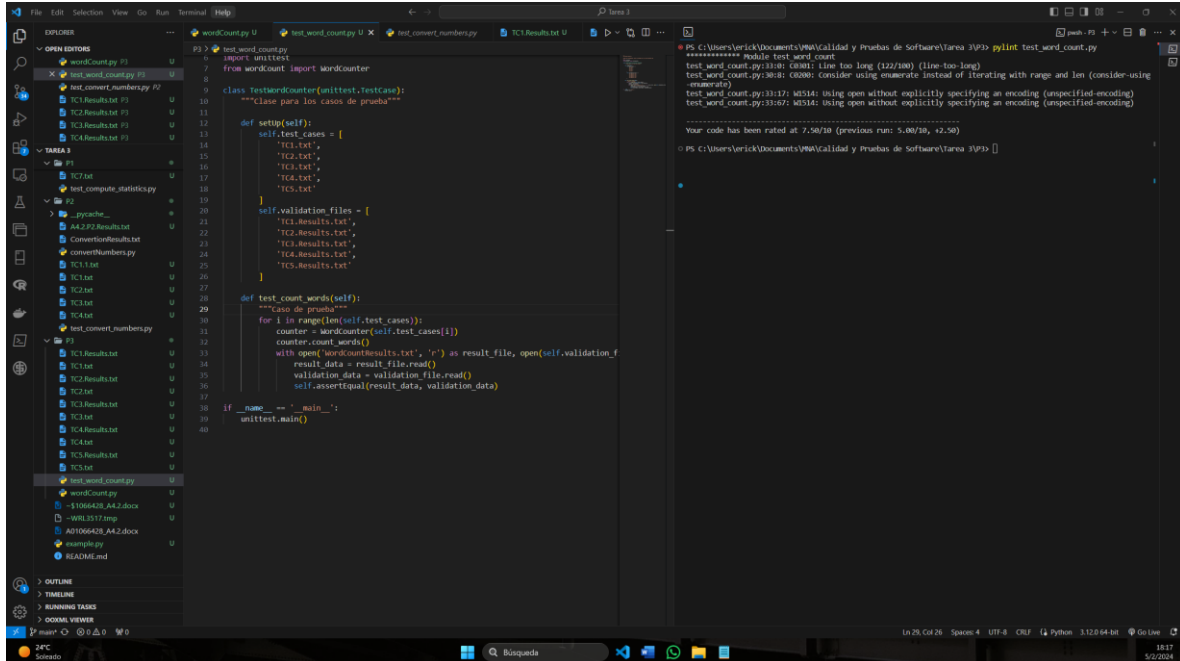
**Nota\*** El archivo de resultados presentaba algunos errores, el archivo de datos TC1.txt no coincide con los datos en el archivo de resultados, los datos manejados como negativos están convertidos con una mascara a 2 bits, 10bits y 32bits, fue más complicado y tardado encontrar estos patrones que completar la actividad.

Problem 3: Count Words	
Description	Requirement 1. The program shall be invoked from a command line. The program shall receive a file as parameter. The file will contain a word (presumable between spaces).
	Requirement 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. <b>All computation MUST be calculated using the basic algorithms, not functions or libraries.</b>
	Requirement 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.
	Requirement 4. The name of the program shall be wordCount.py
	Requirement 5. The minimum format to invoke the program shall be as follows: python wordCount.py fileWithData.txt
	Requirement 6. The program shall manage files having from hundreds of items to thousands of items.
	Requirement 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.
	Requirement 8. Be compliant with PEP8.

## 1. Implementación y corrección del problema 3 y primera revisión usando PyLint:



## 2. Creación un archivo de pruebas unitarias:





### 3. Corrección de código usando PyLint:

