4.2 Ejercicio de programación 1

Pruebas de software y aseguramiento de la calidad

Victor Hugo Vazquez Herrera

A01795624

Problema 1:

Código en python:

Python

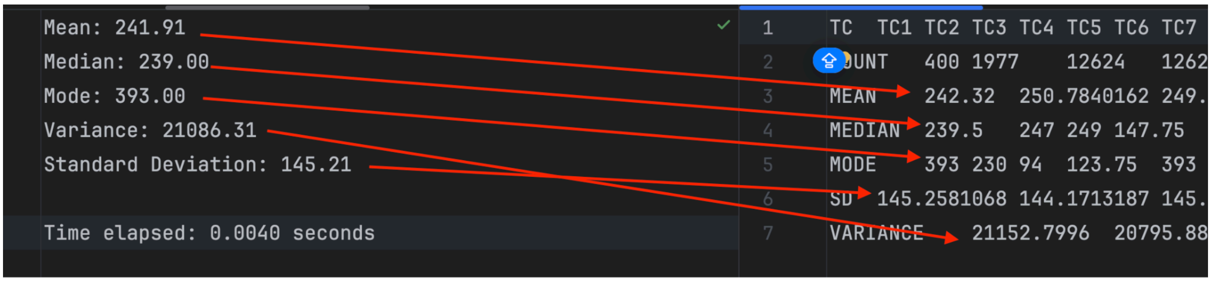
```python
"""
Compute Statistics Script
This Python script reads a file containing a list of numbers, computes
descriptive statistics,
and saves the results to a file. The computations use only basic
algorithms without built-in
statistical functions or libraries.
"""
import sys
import time
def read_numbers(file_path: str):
    """
    Reads numbers from a file and returns a list of valid numbers.
    Args:
        file_path (str): Path to the input file.
    Returns:
        list: A list of valid numbers.
    """
    numbers = []
    try:
        with open(file_path, 'r', encoding="utf-8") as file:
            for line in file:
                for value in line.split():
                    try:
                        numbers.append(float(value))
                    except ValueError:
                        print(f"Warning: Invalid data '{value}'
ignored.")
    except OSError as error:
        print(f"Error reading file: {error}")
        sys.exit(1)
    return numbers
def compute_mean(data: list):
    """Calculates the mean."""
    total = 0
    count = 0
    for num in data:
        total += num
        count += 1
    return total / count if count > 0 else 0
def compute_median(data: list):
    """Calculates the median by sorting and selecting the middle value."""
    n = len(data)
    if n == 0:
        return 0
    sorted_data = data[:]
    for i in range(n - 1):  # We use simple bubble sort
        for j in range(n - i - 1):
            if sorted_data[j] > sorted_data[j + 1]:
                sorted_data[j], sorted_data[j + 1] = sorted_data[j + 1], sorted_data[j]
    mid = n // 2
    if n % 2 == 0:
        return (sorted_data[mid - 1] + sorted_data[mid]) / 2
    return sorted_data[mid]
def compute_mode(data: list):
    """Calculates the mode by counting occurrences."""
    frequency = {}
    max_count = 0
    mode = None
    for num in data:
        frequency[num] = frequency.get(num, 0) + 1
        if frequency[num] > max_count:
            max_count = frequency[num]
            mode = num
    if max_count == 1:
        return "N/A"
    return mode
def compute_variance(data: list, mean: float):
    """Computes variance."""
    if len(data) < 2:
        return 0
    variance_sum = 0
    for num in data:
        variance_sum += (num - mean) ** 2
    return variance_sum / len(data)
def compute_std_dev(variance: float):
    """Computes standard deviation as the square root of variance."""
    if variance == 0:
        return 0
    x = variance
    guess = x / 2
    while True:  # We implement square root using Newton's method
        new_guess = (guess + x / guess) / 2
        if abs(new_guess - guess) < 1e-6:
            return new_guess
        guess = new_guess
def write_results(mean, median, mode, variance, std_dev,
elapsed_time):
    """
    Writes the computed statistics to a file.
    Args:
        mean (float): Computed mean.
        median (float): Computed median.
        mode (float): Computed mode.
        variance (float): Computed variance.
        std_dev (float): Computed standard deviation.
        elapsed_time (float): Execution time.
    """
    try:
        with open('StatisticsResults.txt', 'w', encoding="utf-8") as
result_file:
            result_file.write(f"Mean: {mean:.2f}\n")
            result_file.write(f"Median: {median:.2f}\n")
            if mode == "N/A":
                result_file.write(f"Mode: {mode}\n")
            else:
                result_file.write(f"Mode: {mode:.2f}\n")
            result_file.write(f"Variance: {variance:.2f}\n")
            result_file.write(f"Standard Deviation: {std_dev:.2f}\n")
            result_file.write(f"\nTime elapsed: {elapsed_time:.4f}
seconds\n")
    except OSError as error:
        print(f"Error writing results: {error}")
if __name__ == '__main__':
    if len(sys.argv) != 2:
        print("Usage: python computeStatistics.py
<file_with_data.txt>")
        sys.exit(1)
    input_file = sys.argv[1]
    start_time = time.time()
    log_numbers = read_numbers(input_file)
    if not log_numbers:
        print("Error: No valid numbers found in the file.")
        sys.exit(1)
    log_mean = compute_mean(log_numbers)
    log_median = compute_median(log_numbers)
    log_mode = compute_mode(log_numbers)
    log_variance = compute_variance(log_numbers, log_mean)
    log_std_dev = compute_std_dev(log_variance)
    log_elapsed_time = time.time() - start_time
    print(f"Mean: {log_mean:.2f}")
    print(f"Median: {log_median:.2f}")
    if log_mode == "N/A":
        print(f"Mode: {log_mode}")
    else:
        print(f"Mode: {log_mode:.2f}")
    print(f"Variance: {log_variance:.2f}")
    print(f"Standard Deviation: {log_std_dev:.2f}")
    print(f"\nElapsed time: {log_elapsed_time:.4f} seconds")
    write_results(log_mean, log_median, log_mode, log_variance,
log_std_dev, log_elapsed_time)
```

Comparativa resultados:

En la siguiente captura podemos observar lado a lado los resultados del código (datos de la izquierda) y los proporcionados (archivo results-errata.txt). Podemos observar la gran similitud entre ambos y asimismo las diferencias se pueden atribuir a la manera de redondear de los algoritmos empleados en los códigos de Python.

Inclusive se manejan los casos en que la moda de los datos no existe, pues todos los valores son únicos y también el caso cuando hay elementos que no son números.

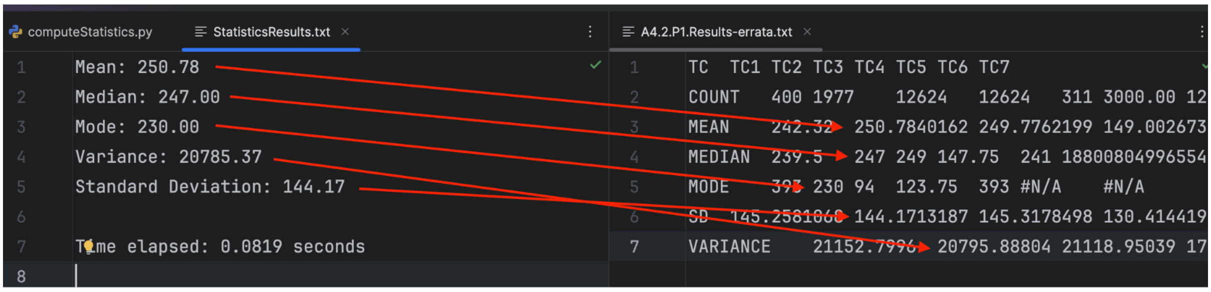
TC1.txt



```
Mean: 241.91
Median: 239.00
Mode: 393.00
Variance: 21086.31
Standard Deviation: 145.21

Time elapsed: 0.0040 seconds
```

```
TC   TC1 TC2 TC3 TC4 TC5 TC6 TC7
 UNT     400 1977    12624   1262
MEAN     242.32  250.7840162 249.
MEDIAN   239.5   247 249 147.75
MODE     393 230 94  123.75  393
SD  145.2581068 144.1713187 145.
VARIANCE    21152.7996  20795.88
```

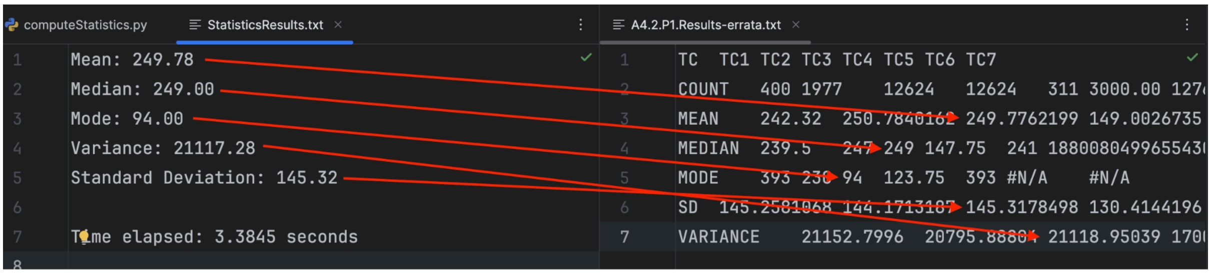
TC2.txt



```
Mean: 250.78
Median: 247.00
Mode: 230.00
Variance: 20785.37
Standard Deviation: 144.17

Time elapsed: 0.0819 seconds
```

```
TC   TC1 TC2 TC3 TC4 TC5 TC6 TC7
COUNT   400 1977     12624    12624    311 3000.00 12
MEAN    242.32  250.7840162 249.7762199 149.002673
MEDIAN  239.5   247 249 147.75   241 18800804996554
MODE    393 230 94  123.75  393 #N/A     #N/A
SD   145.2581068 144.1713187 145.3178498 130.414419
VARIANCE     21152.7996  20795.88804 21118.95039 17
```

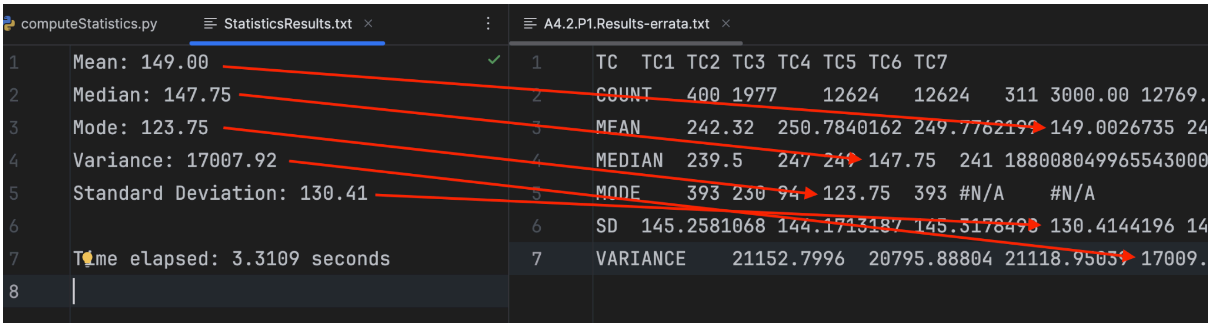
TC3.txt



```
Mean: 249.78
Median: 249.00
Mode: 94.00
Variance: 21117.28
Standard Deviation: 145.32

Time elapsed: 3.3845 seconds
```

```
TC   TC1 TC2 TC3 TC4 TC5 TC6 TC7
COUNT    400 1977     12624    12624    311 3000.00 127
MEAN     242.32  250.7840162 249.7762199 149.0026735
MEDIAN   239.5   247 249 147.75   241 188008049965543
MODE     393 230 94  123.75  393 #N/A     #N/A
SD   145.2581068 144.1713187 145.3178498 130.4144196
VARIANCE     21152.7996  20795.88804 21118.95039 170
```

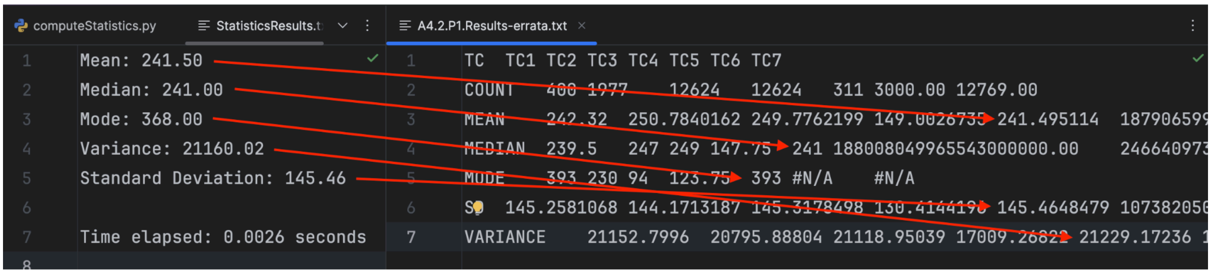
TC4.txt



```
Mean: 149.00
Median: 147.75
Mode: 123.75
Variance: 17007.92
Standard Deviation: 130.41

Time elapsed: 3.3109 seconds
```

```
TC   TC1 TC2 TC3 TC4 TC5 TC6 TC7
COUNT    400 1977     12624    12624    311 3000.00 12769.
MEAN     242.32  250.7840162 249.7762199 149.0026735 24
MEDIAN   239.5   247 249 147.75   241 188008049965543000
MODE     393 230 94  123.75  393 #N/A     #N/A
SD   145.2581068 144.1713187 145.3178498 130.4144196 14
VARIANCE     21152.7996  20795.88804 21118.95039 17009.
```

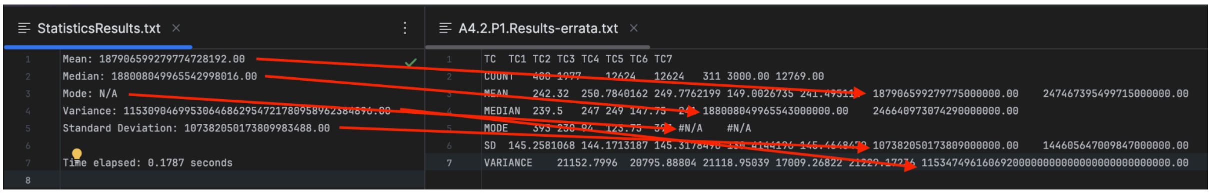
TC5.txt



```
Mean: 241.50
Median: 241.00
Mode: 368.00
Variance: 21160.02
Standard Deviation: 145.46

Time elapsed: 0.0026 seconds
```

```
TC   TC1 TC2 TC3 TC4 TC5 TC6 TC7
COUNT    400 1977     12624    12624    311 3000.00 12769.00
MEAN     242.32  250.7840162 249.7762199 149.0026735 241.495114  187906599
MEDIAN   239.5   247 249 147.75   241 188008049965543000000.00     246640973
MODE     393 230 94  123.75  393 #N/A     #N/A
SD   145.2581068 144.1713187 145.3178498 130.4144196 145.4648479 107382050
VARIANCE     21152.7996  20795.88804 21118.95039 17009.26822 21229.17236 1
```

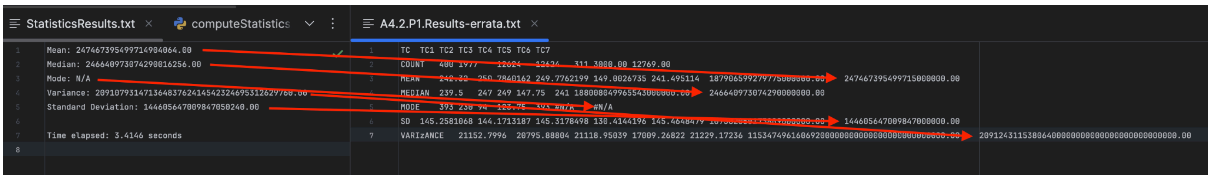
TC6.txt



```
Mean: 187906599279774728192.00
Median: 188008049965542998016.00
Mode: N/A
Variance: 11530904699530646862954721780958962384896.00
Standard Deviation: 107382050173809983488.00

Time elapsed: 0.1787 seconds
```

```
TC   TC1 TC2 TC3 TC4 TC5 TC6 TC7
COUNT    400 1977     12624    12624    311 3000.00 12769.00
MEAN     242.32  250.7840162 249.7762199 149.0026735 241.495114  187906599279775000000.00    247467395499715000000.00
MEDIAN   239.5   247 249 147.75   241 188008049965543000000.00     246640973074290000000.00
MODE     393 230 94  123.75  393 #N/A     #N/A
SD   145.2581068 144.1713187 145.3178498 130.4144196 145.4648479 107382050173809000000.00    144605647009847000000.00
VARIANCE     21152.7996  20795.88804 21118.95039 17009.26822 21229.17236 1153474961606920000000000000000000000000.00
```

TC7.txt

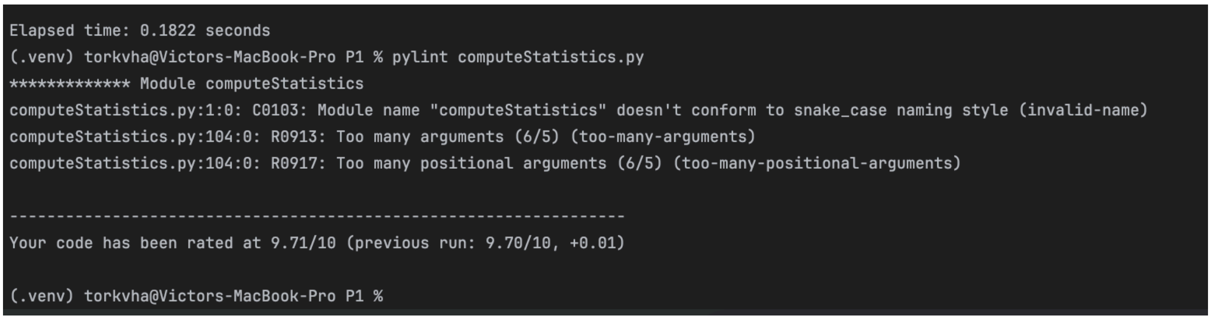


```
Mean: 247467395499714904064.00
Median: 246640973074290016256.00
Mode: N/A
Variance: 20910793147136483762414542324695312629760.00
Standard Deviation: 144605647009847050240.00

Time elapsed: 3.4146 seconds
```

```
TC   TC1 TC2 TC3 TC4 TC5 TC6 TC7
COUNT    400 1977     12624    12624    311 3000.00 12769.00
MEAN     242.32  250.7840162 249.7762199 149.0026735 241.495114  187906599279775000000.00    247467395499715000000.00
MEDIAN   239.5   247 249 147.75   241 188008049965543000000.00     246640973074290000000.00
MODE     393 230 94  123.75  393 #N/A     #N/A
SD   145.2581068 144.1713187 145.3178498 130.4144196 145.4648479 107382050173809000000.00    144605647009847000000.00
VARIzANCE    21152.7996  20795.88804 21118.95039 17009.26822 21229.17236 1153474961606920000000000000000000000000.00  20912431153806400000000000000000000000000.00
```

Resultado de pylint:

En seguida vemos que la calificación del código es bastante alta y los errores mostrados se atribuyen a las características de los requerimientos del problema.



```
Elapsed time: 0.1822 seconds
(.venv) torkvha@Victors-MacBook-Pro P1 % pylint computeStatistics.py
************* Module computeStatistics
computeStatistics.py:1:0: C0103: Module name "computeStatistics" doesn't conform to snake_case naming style (invalid-name)
computeStatistics.py:104:0: R0913: Too many arguments (6/5) (too-many-arguments)
computeStatistics.py:104:0: R0917: Too many positional arguments (6/5) (too-many-positional-arguments)

------------------------------------------------------------------
Your code has been rated at 9.71/10 (previous run: 9.70/10, +0.01)

(.venv) torkvha@Victors-MacBook-Pro P1 %
```

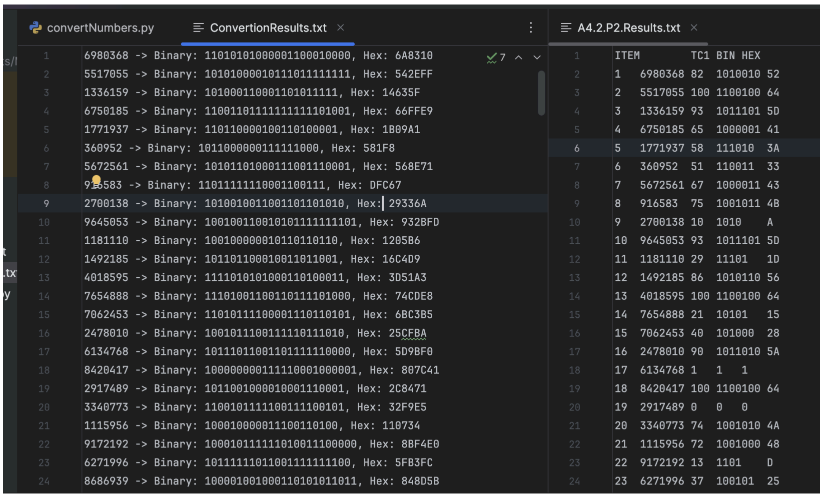
Problema 2:

Código en python:

```python
"""
Converter Program
This Python code reads a file containing numbers, converts each number to binary and hexadecimal
using basic algorithms (without built-in functions), and writes the results to a file.
Errors are handled gracefully, and the execution time is recorded.
Usage:
    python convertNumbers.py fileWithData.txt
"""
import sys
import time
def to_binary(num: int) -> str:
    """Converts a decimal number to binary using basic division-by-2 method."""
    if num == 0:
        return "0"
    binary = ""
    while num > 0:
        binary = str(num % 2) + binary
        num //= 2
    return binary
def to_hexadecimal(num: int) -> str:
    """Converts a decimal number to hexadecimal using basic division-by-16 method."""
    if num == 0:
        return "0"
    hex_digits = "0123456789ABCDEF"
    hexadecimal = ""
    while num > 0:
        remainder = num % 16
        hexadecimal = hex_digits[remainder] + hexadecimal
        num //= 16
    return hexadecimal
def process_file(file_path: str):
    """
    Reads numbers from a file, converts them to binary and hexadecimal,
    and writes the results to 'ConvertionResults.txt'.
    Invalid data is logged as an error and skipped.
    """
    results = []
    try:
        with open(file_path, "r", encoding="utf-8") as file:
            for line_num, line in enumerate(file, start=1):
                line = line.strip()
                if not line:
                    continue  # Skip empty lines
                try:
                    number = int(line)
                    binary = to_binary(number)
                    hexadecimal = to_hexadecimal(number)
                    results.append(f"{number} -> Binary: {binary}, Hex: {hexadecimal}")
                except ValueError:
                    print(f"Error: Invalid number on line {line_num}: '{line}'")
    except OSError as error:
        print(f"Error reading file: {error}")
        return []
    # Write results to file
    try:
        with open("ConvertionResults.txt", "w", encoding="utf-8") as result_file:
            for result in results:
                result_file.write(result + "\n")
    except OSError as error:
        print(f"Error writing results: {error}")
        return []
    return results
if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Usage: python convertNumbers.py <file_with_data.txt>")
        sys.exit(1)
    input_file = sys.argv[1]
    start_time = time.time()
    output_results = process_file(input_file)
    elapsed_time = time.time() - start_time
    if output_results:
        for output in output_results:
            print(output)
    print(f"\nExecution Time: {elapsed_time:.4f} seconds")
    try:
        with open("ConvertionResults.txt", "a", encoding="utf-8") as
log_result_file:
            log_result_file.write(f"\nExecution Time:
{elapsed_time:.4f} seconds\n")
    except OSError as error:
        print(f"Error appending execution time: {error}")
```

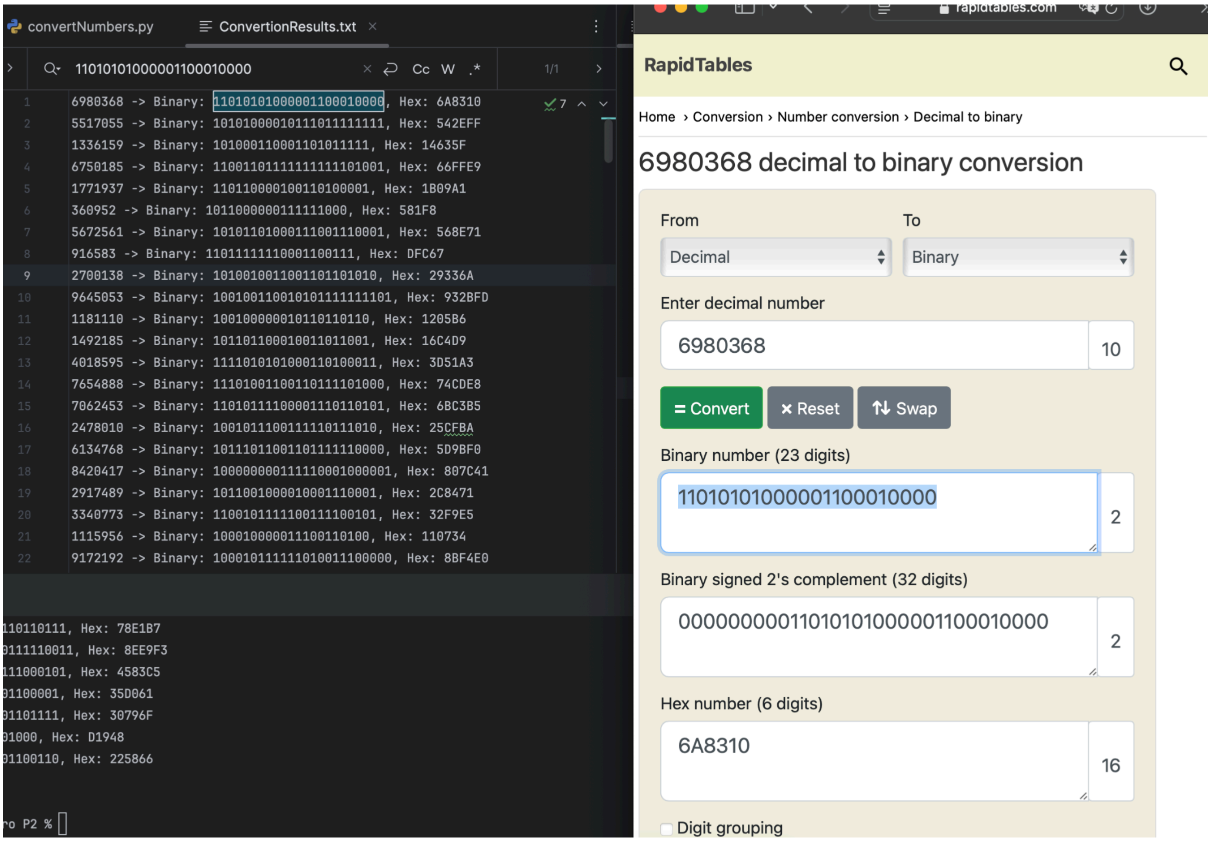
Comparativa resultados:

A primera instancia podemos observar que los resultados obtenidos discrepan del archivo txt de resultados proporcionado. No obstante haciendo una búsqueda en Google para validar los datos, vemos que los resultados obtenidos coinciden con los que se generan en mi código, por lo que asumimos que no se puede comparar y se deberian de generar algún tipo de pruebas mas duras para que este tipo de errores no pasen desapercibidos.



```
6980368 -> Binary: 11010101000001100010000, Hex: 6A8310
5517055 -> Binary: 10101000010111011111111, Hex: 542EFF
1336159 -> Binary: 101000110001101011111, Hex: 14635F
6750185 -> Binary: 11001101111111111101001, Hex: 66FFE9
1771937 -> Binary: 110110000100110100001, Hex: 1B09A1
360952 -> Binary: 1011000000111111000, Hex: 581F8
5672561 -> Binary: 10101101000111001110001, Hex: 568E71
916583 -> Binary: 11011111110001100111, Hex: DFC67
2700138 -> Binary: 1010010011001101101010, Hex: 29336A
9645053 -> Binary: 100100110010101111111101, Hex: 932BFD
1181110 -> Binary: 100100000010110110110, Hex: 1205B6
1492185 -> Binary: 101101100010011011001, Hex: 16C4D9
4018595 -> Binary: 1111010101000110100011, Hex: 3D51A3
7654888 -> Binary: 11101001100110111101000, Hex: 74CDE8
7062453 -> Binary: 11010111100001110110101, Hex: 6BC3B5
2478010 -> Binary: 1001011100111110111010, Hex: 25CFBA
6134768 -> Binary: 10111011001101111110000, Hex: 5D9BF0
8420417 -> Binary: 100000000111110001000001, Hex: 807C41
2917489 -> Binary: 1011001000010001110001, Hex: 2C8471
3340773 -> Binary: 1100101111100111100101, Hex: 32F9E5
1115956 -> Binary: 100010000011100110100, Hex: 110734
9172192 -> Binary: 100010111111010011100000, Hex: 8BF4E0
6271996 -> Binary: 10111111011001111111100, Hex: 5FB3FC
8686939 -> Binary: 100001001000110101011011, Hex: 848D5B
```

```
ITEM        TC1 BIN HEX
1   6980368 82  1010010 52
2   5517055 100 1100100 64
3   1336159 93  1011101 5D
4   6750185 65  1000001 41
5   1771937 58  111010  3A
6   360952  51  110011  33
7   5672561 67  1000011 43
8   916583  75  1001011 4B
9   2700138 10  1010    A
10  9645053 93  1011101 5D
11  1181110 29  11101   1D
12  1492185 86  1010110 56
13  4018595 100 1100100 64
14  7654888 21  10101   15
15  7062453 40  101000  28
16  2478010 90  1011010 5A
17  6134768 1   1   1
18  8420417 100 1100100 64
19  2917489 0   0   0
20  3340773 74  1001010 4A
21  1115956 72  1001000 48
22  9172192 13  1101    D
23  6271996 37  100101  25
```



```
11010101000001100010000
6980368 -> Binary: 11010101000001100010000, Hex: 6A8310
5517055 -> Binary: 10101000010111011111111, Hex: 542EFF
1336159 -> Binary: 101000110001101011111, Hex: 14635F
6750185 -> Binary: 11001101111111111101001, Hex: 66FFE9
1771937 -> Binary: 110110000100110100001, Hex: 1B09A1
360952 -> Binary: 1011000000111111000, Hex: 581F8
5672561 -> Binary: 10101101000111001110001, Hex: 568E71
916583 -> Binary: 11011111110001100111, Hex: DFC67
2700138 -> Binary: 1010010011001101101010, Hex: 29336A
9645053 -> Binary: 100100110010101111111101, Hex: 932BFD
1181110 -> Binary: 100100000010110110110, Hex: 1205B6
1492185 -> Binary: 101101100010011011001, Hex: 16C4D9
4018595 -> Binary: 1111010101000110100011, Hex: 3D51A3
7654888 -> Binary: 11101001100110111101000, Hex: 74CDE8
7062453 -> Binary: 11010111100001110110101, Hex: 6BC3B5
2478010 -> Binary: 1001011100111110111010, Hex: 25CFBA
6134768 -> Binary: 10111011001101111110000, Hex: 5D9BF0
8420417 -> Binary: 100000000111110001000001, Hex: 807C41
2917489 -> Binary: 1011001000010001110001, Hex: 2C8471
3340773 -> Binary: 1100101111100111100101, Hex: 32F9E5
1115956 -> Binary: 100010000011100110100, Hex: 110734
9172192 -> Binary: 100010111111010011100000, Hex: 8BF4E0
```

```
110110111, Hex: 78E1B7
0111110011, Hex: 8EE9F3
111000101, Hex: 4583C5
01100001, Hex: 35D061
01101111, Hex: 30796F
01000, Hex: D1948
01100110, Hex: 225866

ro P2 %
```

Home › Conversion › Number conversion › Decimal to binary

6980368 decimal to binary conversion

From: Decimal

To: Binary

Enter decimal number: 6980368 (10)

= Convert × Reset ⇅ Swap

Binary number (23 digits): 11010101000001100010000 (2)

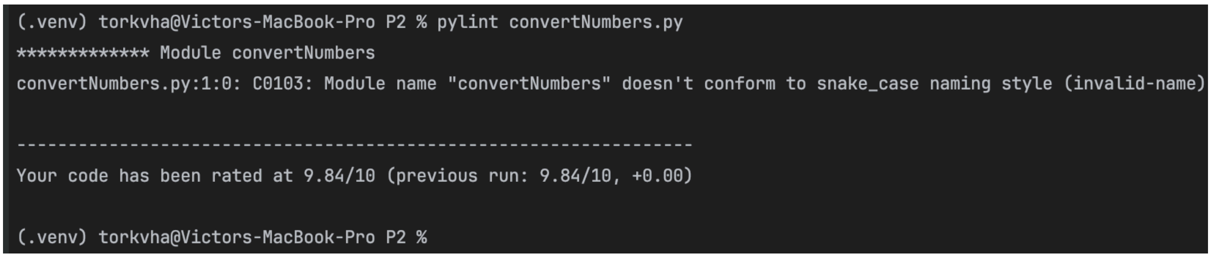
Binary signed 2's complement (32 digits): 00000000011010101000001100010000 (2)

Hex number (6 digits): 6A8310 (16)

□ Digit grouping

Resultado de pylint:

Podemos observar que la única recomendación es cambiar el nombre del archivo. No obstante, nuevamente se trata de un requerimiento.



```
(.venv) torkvha@Victors-MacBook-Pro P2 % pylint convertNumbers.py
************* Module convertNumbers
convertNumbers.py:1:0: C0103: Module name "convertNumbers" doesn't conform to snake_case naming style (invalid-name)

------------------------------------------------------------------
Your code has been rated at 9.84/10 (previous run: 9.84/10, +0.00)

(.venv) torkvha@Victors-MacBook-Pro P2 %
```

Problema 3:

Código en python:

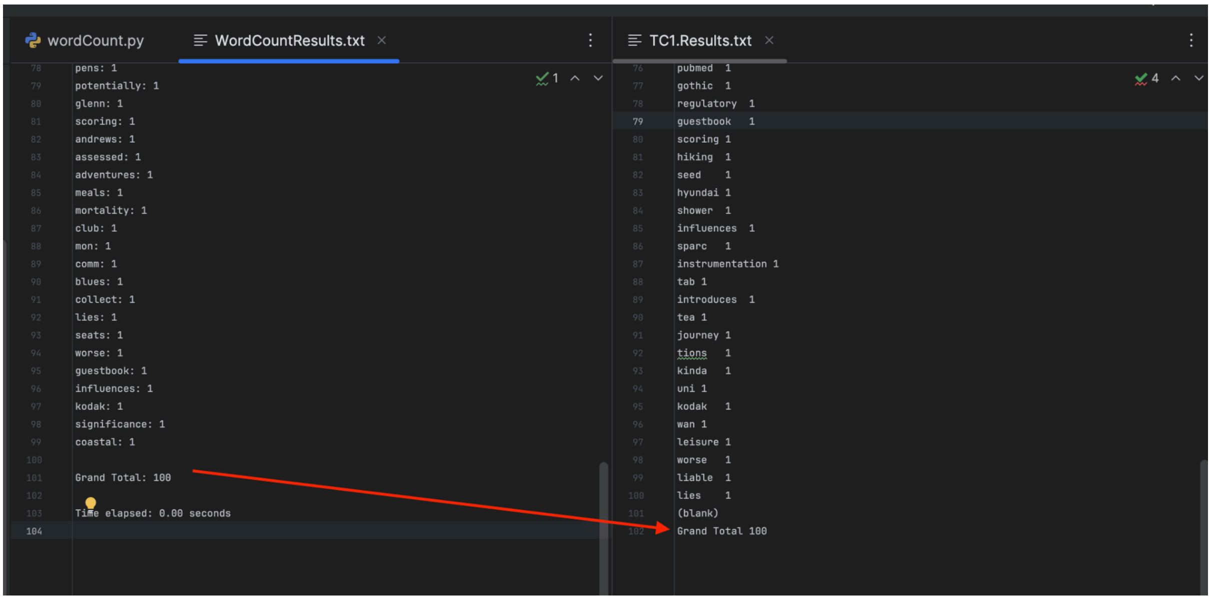
```python
"""
Word Count Script
```

```
This Python script reads a file, counts word occurrences, and saves
the results to a file.
"""
import sys
import time
def count_words(file_path):
    """
    Counts occurrences of words in a given text file without using
built-in string functions.
    Args:
        file_path (str): The path to the file to be read.
    Returns:
        dict: A dictionary containing words as keys and their
frequencies as values.
    """
    word_frequencies = {}
    try:
        with open(file_path, 'r', encoding="utf-8") as file:
            for line in file:
                words = line.split()
                for raw_word in words:
                    cleaned_word = ''.join(
                        char.lower() if 'A' <= char <= 'Z' or 'a' <=
char <= 'z' else ''
                        for char in raw_word
                    )
                    if cleaned_word:
                        word_frequencies[cleaned_word] =
word_frequencies.get(cleaned_word, 0) + 1
    except OSError as error:
        print(f"Error reading file: {error}")
    return word_frequencies
def write_results(results, elapsed_time):
    """
    Writes word count results and execution time to a file.
    Args:
        results (dict): A dictionary containing word frequencies.
        elapsed_time (float): Execution time.
    """
    try:
        with open('WordCountResults.txt', 'w', encoding="utf-8") as
result_file:
            total_count = 0
            for word, count in results.items():
                result_file.write(f"{word}: {count}\n")
                total_count += count
            result_file.write(f"\nGrand Total: {total_count}\n")
            result_file.write(f"\nTime elapsed: {elapsed_time:.2f}
seconds\n")
    except OSError as error:
        print(f"Error writing results: {error}")
if __name__ == '__main__':
    if len(sys.argv) != 2:
        print("Usage: python wordCount.py <file_with_data.txt>")
        sys.exit(1)
    input_file = sys.argv[1]
    start_time = time.time()
    word_counts = count_words(input_file)
    log_elapsed_time = time.time() - start_time
    for log_word, log_count in word_counts.items():
        print(f"{log_word}: {log_count}")
    print(f"\nGrand Total: {sum(word_counts.values())}")
    print(f"\nElapsed time: {log_elapsed_time:.4f} seconds")
    write_results(word_counts, log_elapsed_time)
```

Comparativa resultados:

En la siguiente comparativa vemos a simple vista que para cada uno de los escenarios se iguala con perfección el total de los conteos de cada una de las palabras, asimismo si se hace una exploración aleatoria se ve la coincidencia de los resultados.
A continuación, podemos observar cada una de los escenarios y su comparación:

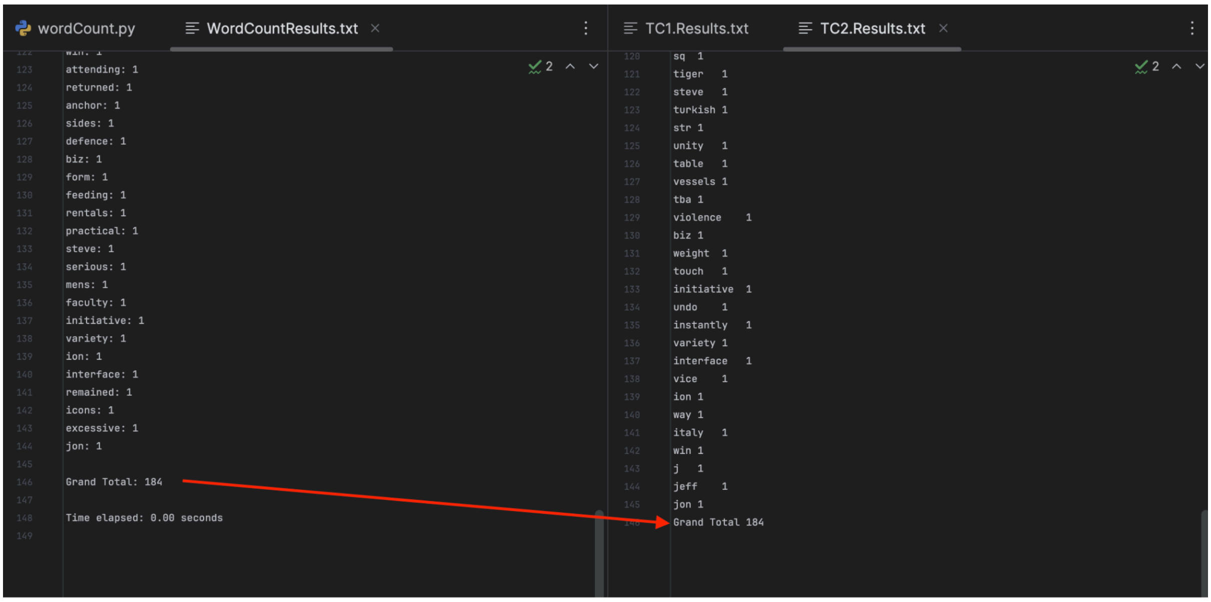
TC1.txt



```
pens: 1
potentially: 1
glenn: 1
scoring: 1
andrews: 1
assessed: 1
adventures: 1
meals: 1
mortality: 1
club: 1
mon: 1
comm: 1
blues: 1
collect: 1
lies: 1
seats: 1
worse: 1
guestbook: 1
influences: 1
kodak: 1
significance: 1
coastal: 1

Grand Total: 100

Time elapsed: 0.00 seconds
```

```
pubmed  1
gothic  1
regulatory  1
guestbook   1
scoring 1
hiking  1
seed    1
hyundai 1
shower  1
influences  1
sparc   1
instrumentation 1
tab 1
introduces  1
tea 1
journey 1
tions   1
kinda   1
uni 1
kodak   1
wan 1
leisure 1
worse   1
liable  1
lies    1
(blank)
Grand Total 100
```

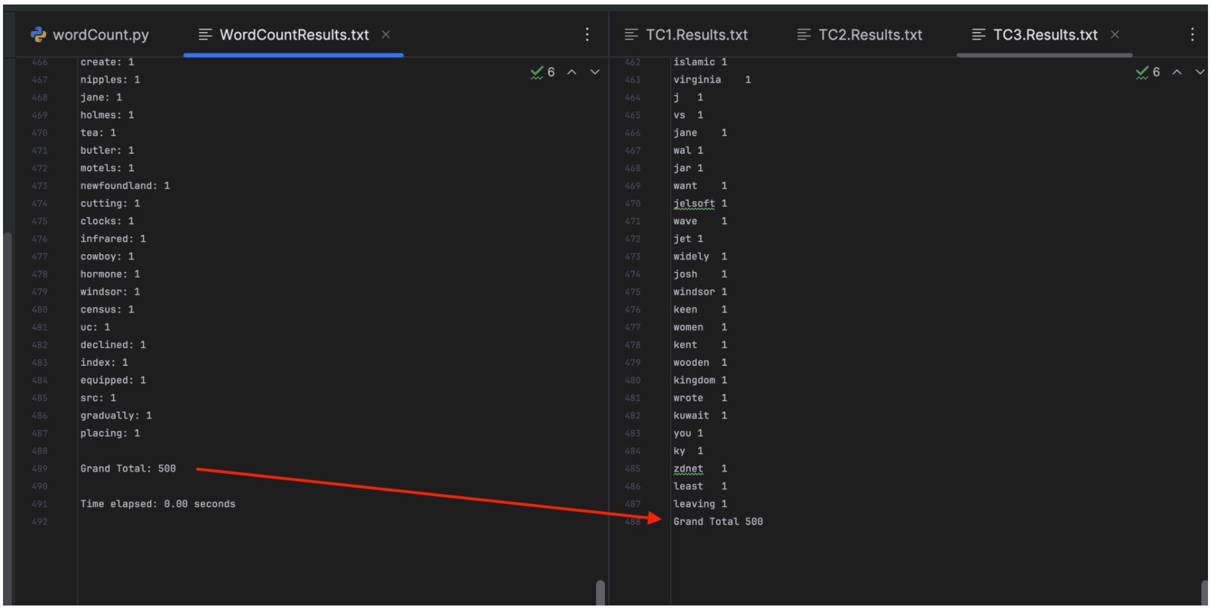
TC2.txt



```
win: 1
attending: 1
returned: 1
anchor: 1
sides: 1
defence: 1
biz: 1
form: 1
feeding: 1
rentals: 1
practical: 1
steve: 1
serious: 1
mens: 1
faculty: 1
initiative: 1
variety: 1
ion: 1
interface: 1
remained: 1
icons: 1
excessive: 1
jon: 1

Grand Total: 184

Time elapsed: 0.00 seconds
```

```
sq  1
tiger   1
steve   1
turkish 1
str 1
unity   1
table   1
vessels 1
tba 1
violence    1
biz 1
weight  1
touch   1
initiative  1
undo    1
instantly   1
variety 1
interface   1
vice    1
ion 1
way 1
italy   1
win 1
j   1
jeff    1
jon 1
Grand Total 184
```

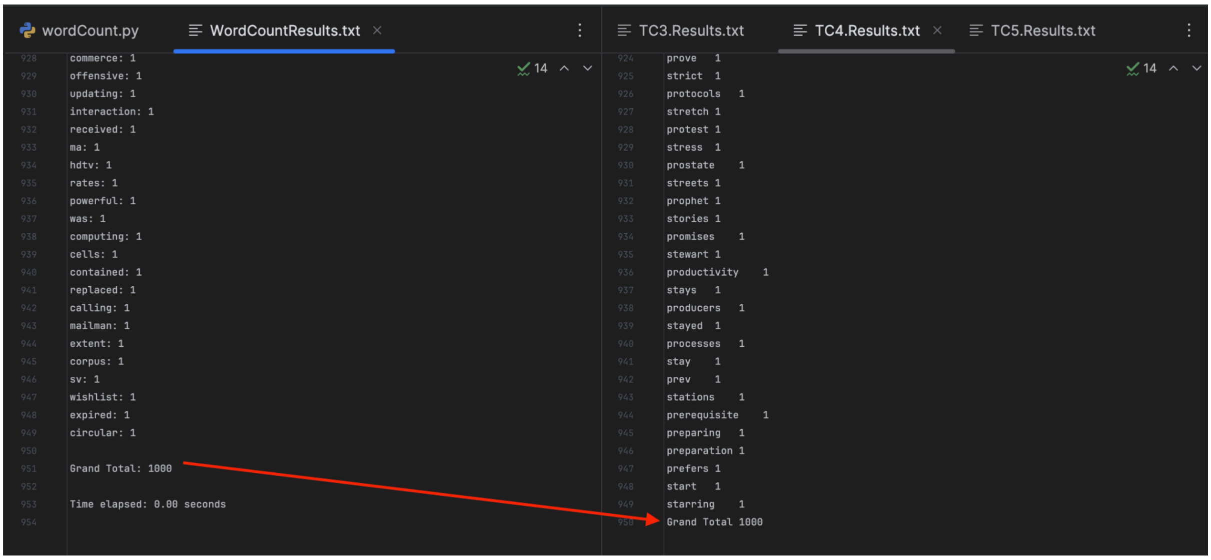
TC3.txt



```
create: 1
nipples: 1
jane: 1
holmes: 1
tea: 1
butler: 1
motels: 1
newfoundland: 1
cutting: 1
clocks: 1
infrared: 1
cowboy: 1
hormone: 1
windsor: 1
census: 1
uc: 1
declined: 1
index: 1
equipped: 1
src: 1
gradually: 1
placing: 1

Grand Total: 500

Time elapsed: 0.00 seconds
```

```
islamic 1
virginia    1
j   1
vs  1
jane    1
wal 1
jar 1
want    1
jelsoft 1
wave    1
jet 1
widely  1
josh    1
windsor 1
keen    1
women   1
kent    1
wooden  1
kingdom 1
wrote   1
kuwait  1
you 1
ky  1
zdnet   1
least   1
leaving 1
Grand Total 500
```

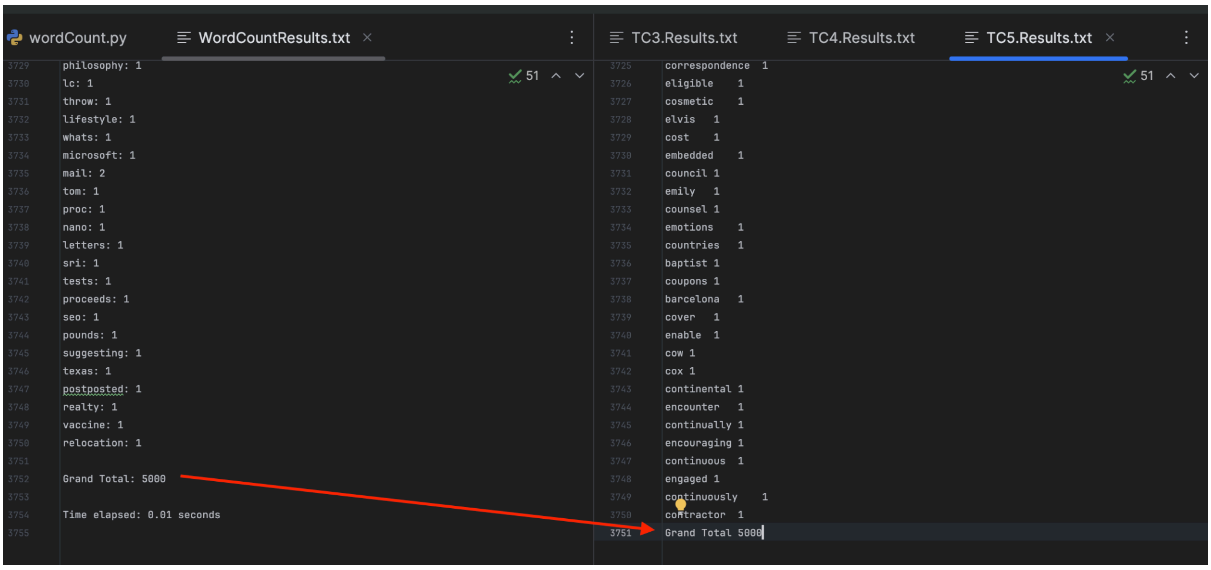
TC4.txt



```
commerce: 1
offensive: 1
updating: 1
interaction: 1
received: 1
ma: 1
hdtv: 1
rates: 1
powerful: 1
was: 1
computing: 1
cells: 1
contained: 1
replaced: 1
calling: 1
mailman: 1
extent: 1
corpus: 1
sv: 1
wishlist: 1
expired: 1
circular: 1

Grand Total: 1000

Time elapsed: 0.00 seconds
```

```
prove   1
strict  1
protocols   1
stretch 1
protest 1
stress  1
prostate    1
streets 1
prophet 1
stories 1
promises    1
stewart 1
productivity    1
stays   1
producers   1
stayed  1
processes   1
stay    1
prev    1
stations    1
prerequisite    1
preparing   1
preparation 1
prefers 1
start   1
starring    1
Grand Total 1000
```

TC5.txt

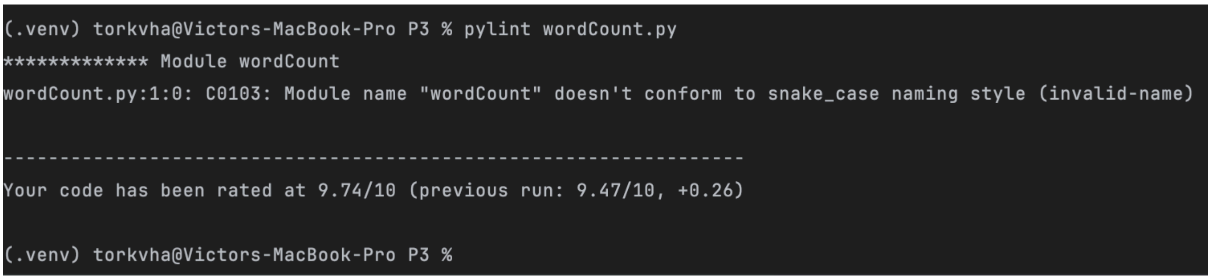


```
philosophy: 1
lc: 1
throw: 1
lifestyle: 1
whats: 1
microsoft: 1
mail: 2
tom: 1
proc: 1
nano: 1
letters: 1
sri: 1
tests: 1
proceeds: 1
seo: 1
pounds: 1
suggesting: 1
texas: 1
postposted: 1
realty: 1
vaccine: 1
relocation: 1

Grand Total: 5000

Time elapsed: 0.01 seconds
```

```
correspondence  1
eligible    1
cosmetic    1
elvis   1
cost    1
embedded    1
council 1
emily   1
counsel 1
emotions    1
countries   1
baptist 1
coupons 1
barcelona   1
cover   1
enable  1
cow 1
cox 1
continental 1
encounter   1
continually 1
encouraging 1
continuous  1
engaged 1
continuously    1
contractor  1
Grand Total 5000
```

Resultado de pylint:

En este problema podemos observar que el único mensaje es respecto al nombre del archivo. Sin embargo, es parte del requerimiento y no lo podemos cambiar.



```
(.venv) torkvha@Victors-MacBook-Pro P3 % pylint wordCount.py
************* Module wordCount
wordCount.py:1:0: C0103: Module name "wordCount" doesn't conform to snake_case naming style (invalid-name)

------------------------------------------------------------------
Your code has been rated at 9.74/10 (previous run: 9.47/10, +0.26)

(.venv) torkvha@Victors-MacBook-Pro P3 %
```