Alejandro Juárez Corona

A01168444

Pruebas de software y aseguramiento de la calidad

Actividad 4.2: Ejercicio de programación

Ejercicio 1. Estadísticas descriptivas

Código:

"""

Module to compute statistics from a file.

"""

import sys

import time

import statistics

def compute\_statistics(file\_path):

"""

Compute decriptive statistics from a file.

- param FILE\_PATH: Path to the file containing data.

"""

data = []

try:

with open(file\_path, 'r', encoding='utf-8') as file:

for line in file:

line = line.strip()

if line:

try:

data.append(float(line))

except ValueError:

print(f"Warning: Skipping invalid entry: {line}")

if not data:

print("No valid data.")

return

except FileNotFoundError:

print(f"Error: File '{file\_path}' not found.")

mean\_value = sum(data) / len(data)

median\_value = statistics.median(data)

mode\_value = statistics.mode(data) if len(data) > 0 else None

stdev\_value = statistics.stdev(data) if len(data) > 1 else None

variance\_value = statistics.variance(data) if len(data) > 1 else None

print(f"Mean: {mean\_value}")

print(f"Median: {median\_value}")

print(f"Mode: {mode\_value}")

print(f"Standard Deviation: {stdev\_value}")

print(f"Variance: {variance\_value}")

with open('StatisticsResults.txt', 'a', encoding='utf-8') as results\_file:

results\_file.write(f"Mean: {mean\_value}\n")

results\_file.write(f"Median: {median\_value}\n")

results\_file.write(f"Mode: {mode\_value}\n")

results\_file.write(f"Standard Deviation: {stdev\_value}\n")

results\_file.write(f"Variance: {variance\_value}\n")

if \_\_name\_\_ == "\_\_main\_\_":

start\_time = time.time()

if len(sys.argv) != 2:

print("Path to data file not in call.")

else:

FILE\_PATH = sys.argv[1]

compute\_statistics(FILE\_PATH)

end\_time = time.time()

elapsed\_time = end\_time - start\_time

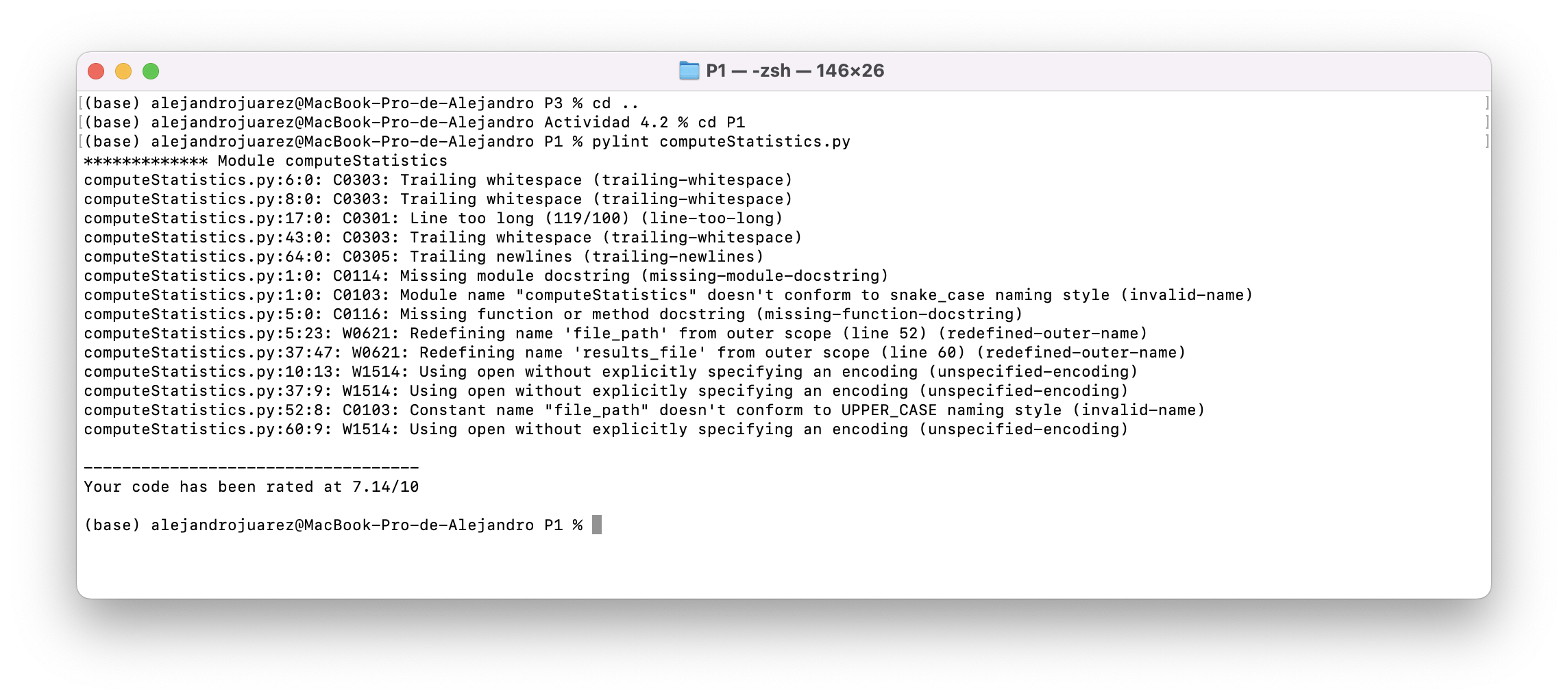
print(f"Execution Time: {elapsed\_time}s")

with open('StatisticsResults.txt', 'a', encoding='utf-8') as results\_file\_append:

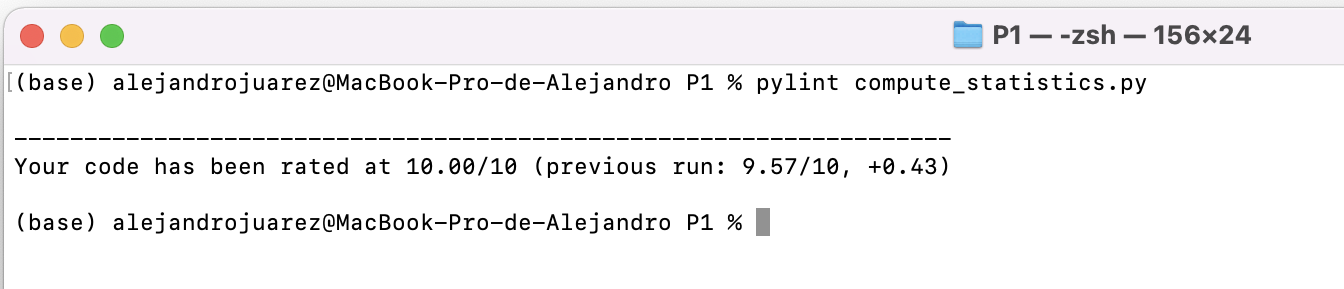
results\_file\_append.write(f"Execution Time for {FILE\_PATH}: {elapsed\_time}s\n")

results\_file\_append.write("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n")

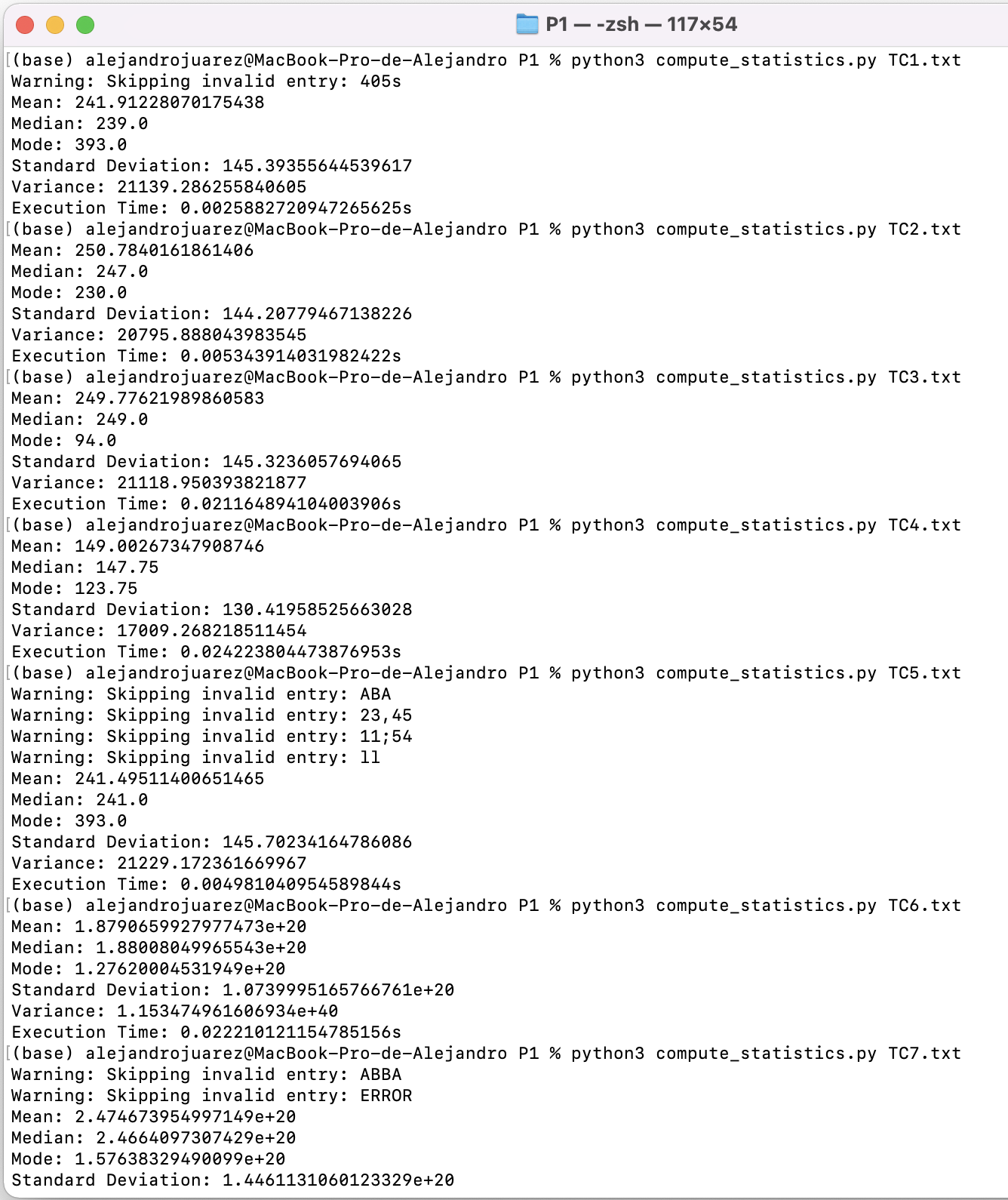
Revisión inicial con Pylint



Después de corregir errores de Pylint:



Resultados de la ejecución:



Ejercicio 2. Conversión de números

Código:

"""

This script reads numbers from a file,

converts them to binary and hexadecimal,

and prints the results.

"""

import sys

import time

def convert\_and\_print\_results(input\_data):

"""

Convert numbers to binary and hexadecimal and print the results.

Parameters:

- data: List of integers to be converted.

Returns:

- tuple: Lists of binary values and hexadecimal values.

"""

bin\_values = []

hex\_values = []

for input\_number in input\_data:

# Convert number to binary

bin\_values.append(bin(input\_number)[2:])

# Convert number to hexadecimal

hex\_values.append(hex(input\_number)[2:].upper())

return bin\_values, hex\_values

if \_\_name\_\_ == "\_\_main\_\_":

start\_time = time.time()

if len(sys.argv) != 2:

print("Path to data file not in call.")

else:

FILE\_PATH = sys.argv[1]

data = []

try:

with open(FILE\_PATH, 'r', encoding='utf-8') as file:

for line in file:

line = line.strip()

if line:

try:

data.append(int(line))

except ValueError:

print(f"Warning: Skipping invalid entry: {line}")

if not data:

print("No valid data to convert.")

else:

# Convert and print results

binary\_values, hexadecimal\_values = convert\_and\_print\_results(data)

# Save results to file

with open('ConversionResults.txt', 'a', encoding='utf-8') as results\_file:

for i, number in enumerate(data):

result = f"N:{number}, B:{binary\_values[i]}, Hex:{hexadecimal\_values[i]}\n"

results\_file.write(result)

except FileNotFoundError:

print(f"Error: File '{FILE\_PATH}' not found.")

end\_time = time.time()

elapsed\_time = end\_time - start\_time

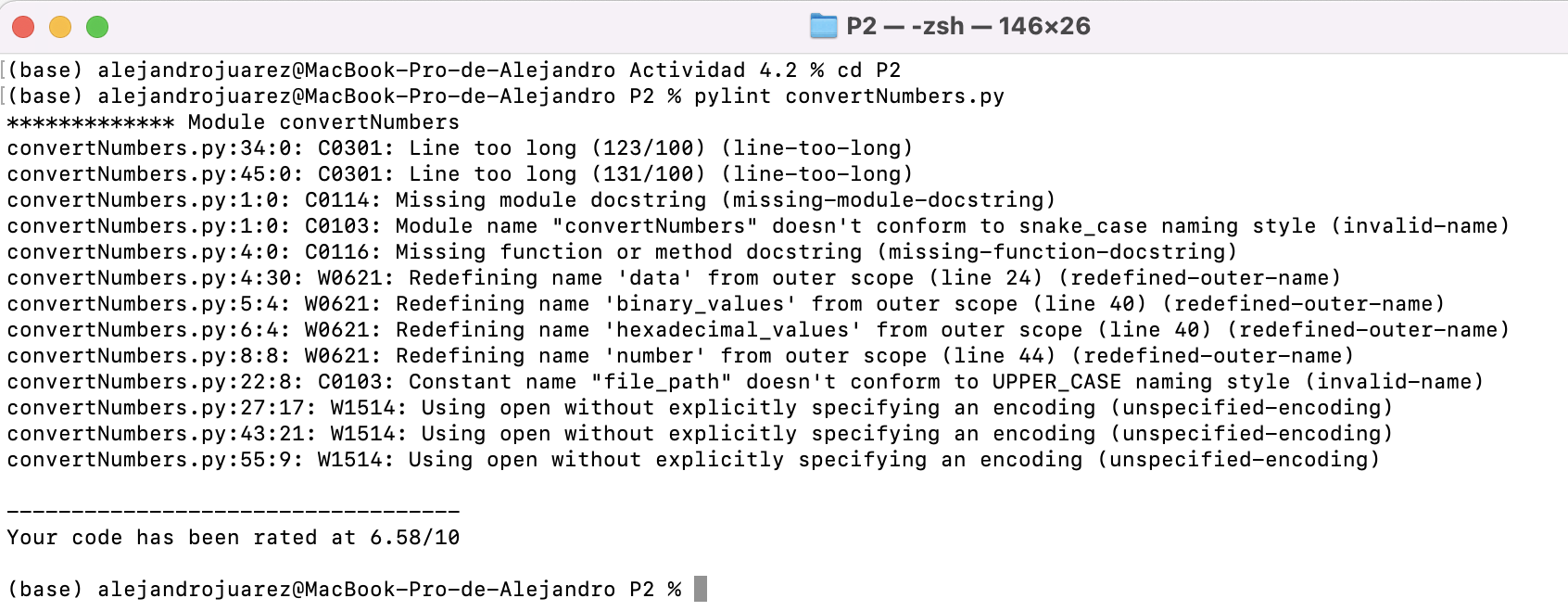
print(f"Execution Time: {elapsed\_time}s")

with open('ConversionResults.txt', 'a', encoding='utf-8') as results\_file:

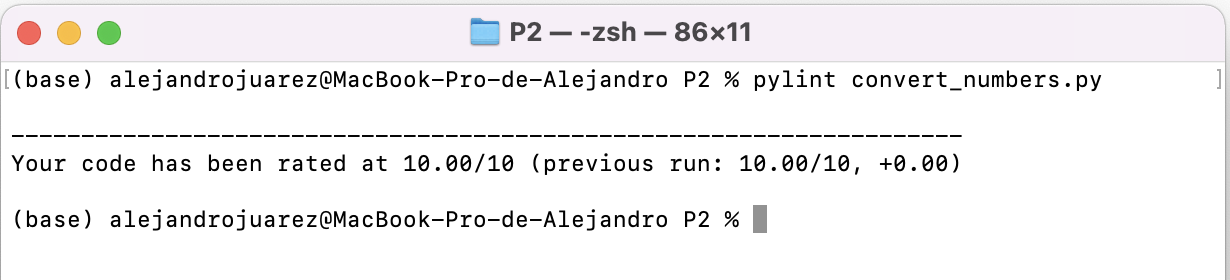
results\_file.write(f"\nExecution Time for {FILE\_PATH}: {elapsed\_time}s\n")

results\_file.write("\n\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n")

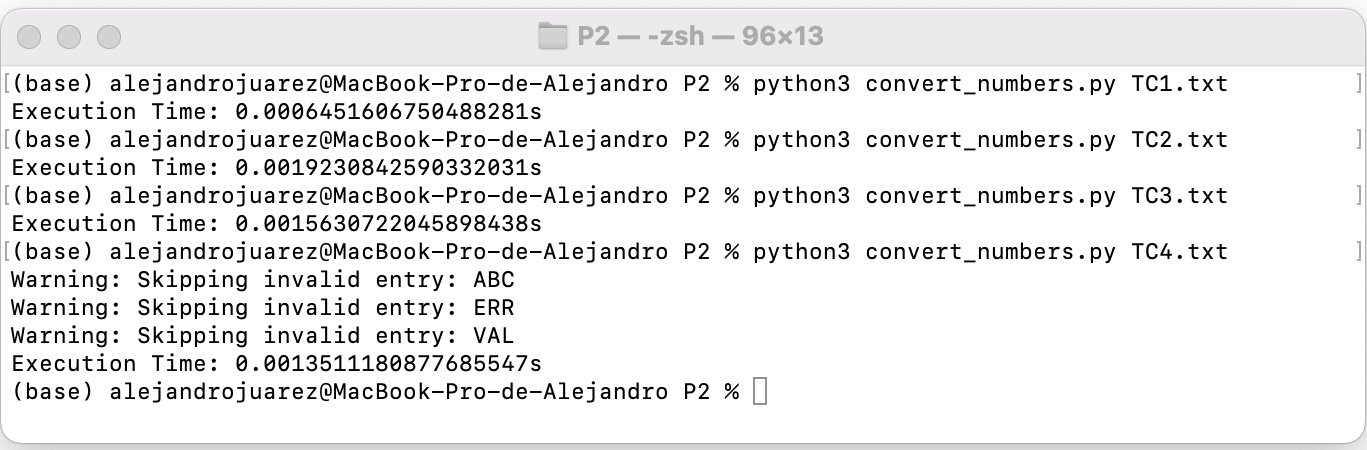
Revisión inicial con Pylint:



Después de corregir errores de Pylint:



Resultados de la ejecución:



Ejercicio 3. Conteo de palabras

Código:

"""Count and print word frequency in a text file."""

import sys

import time

def count\_and\_print\_word\_frequency(file\_path):

"""

Count and print the word frequency in the specified text file.

Parameters:

- file\_path: Path to the text file.

"""

start\_time = time.time()

word\_frequency = {}

try:

with open(file\_path, 'r', encoding='utf-8') as file:

for line in file:

words = line.strip().split()

for word in words:

# Update word frequency

word\_frequency[word] = word\_frequency.get(word, 0) + 1

except FileNotFoundError:

print(f"Error: File '{file\_path}' not found.")

return

# Print results on the screen

print("Word Frequency:")

for word, frequency in word\_frequency.items():

print(f"{word}: {frequency}")

# Save results to file

with open('WordCountResults.txt', 'a', encoding='utf-8') as results\_file:

results\_file.write("Word Frequency:\n")

for word, frequency in word\_frequency.items():

results\_file.write(f"{word}: {frequency}\n")

end\_time = time.time()

elapsed\_time = end\_time - start\_time

# Print and save elapsed time

print(f"\nExecution Time: {elapsed\_time}s")

with open('WordCountResults.txt', 'a', encoding='utf-8') as results\_file:

results\_file.write(f"\nExecution Time: {elapsed\_time}s\n")

results\_file.write("\n\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n")

if \_\_name\_\_ == "\_\_main\_\_":

if len(sys.argv) != 2:

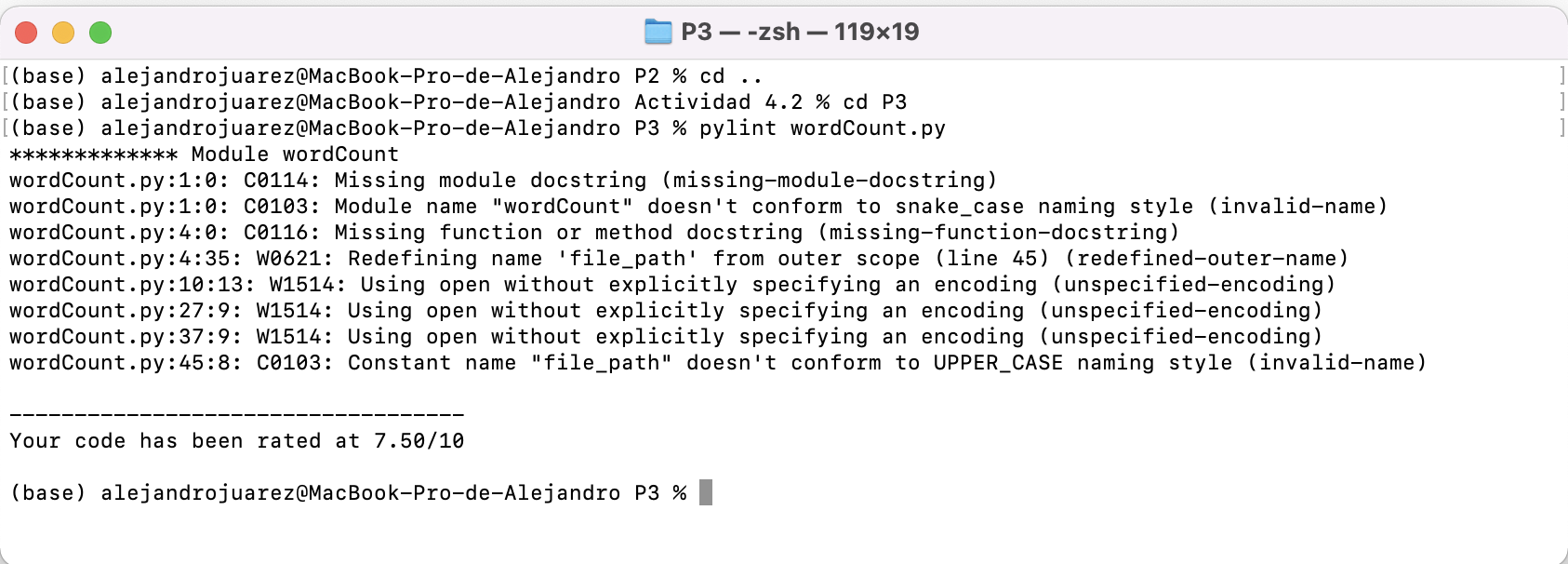
print("Path to data file not in call.")

else:

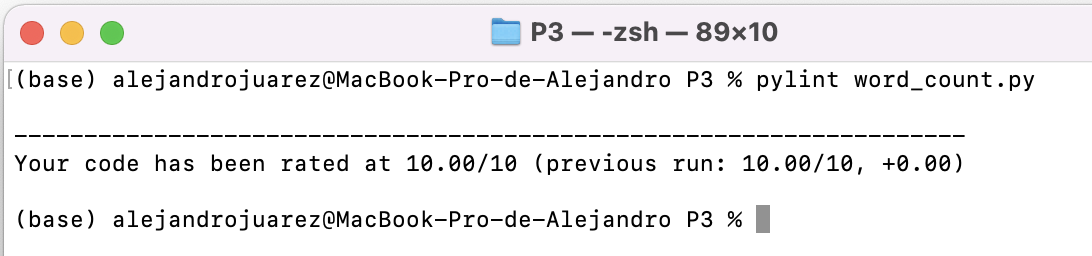
FILE\_PATH = sys.argv[1]

count\_and\_print\_word\_frequency(FILE\_PATH)

Revision inicial con Pylint:



Después de correción de errores:



Resultados de ejecución:

