Actividad 4.2 — Ejercicios de Programación

Alumno: Artemio Santiago Padilla Robles
Matrícula: A01796613
Repositorio: github.com/ArtemioPadilla/good-programming-practices

Descripción

Se implementaron tres programas en Python para procesar archivos de datos, siguiendo el estándar de codificación PEP-8 y validando con `pylint` (score 10.00/10 en los tres). Cada programa se ejecutó contra los casos de prueba proporcionados y se documentaron los resultados.

Programas implementados

P1 — `computeStatistics.py`

Calcula estadísticas descriptivas a partir de un archivo de números.

Requisito	Implementación
Req 1. Invocación por línea de comandos	`python computeStatistics.py archivo.txt`
Req 2. Estadísticas: media, mediana, moda, SD, varianza	Calculadas con algoritmos básicos, sin librerías
Req 3. Manejo de datos inválidos	Se reportan en consola y la ejecución continúa
Req 4. Nombre del programa	`computeStatistics.py`
Req 5. Formato de invocación	`python computeStatistics.py fileWithData.txt`

Requisito	Implementación
Req 6. Escala: cientos a miles	Probado con archivos de hasta 12,769 elementos
Req 7. Tiempo de ejecución	Se muestra en pantalla y en `StatisticsResults.txt`
Req 8. PEP-8	pylint 10.00/10

Algoritmos utilizados:

- **Media**: suma acumulada / N
- **Mediana**: ordenamiento + selección del punto medio
- **Moda**: conteo de frecuencias con diccionario
- **Varianza**: suma de cuadrados de diferencias con denominador N-1
- **Desviación estándar**: raíz cuadrada (método de Newton) con denominador N

Ejemplo de ejecución (TC1):

```
COUNT: 400
MEAN: 242.32
MEDIAN: 239.5
MODE: 393.0
SD: 145.25810683056557
VARIANCE: 21152.79959899749
Elapsed Time: 0.001287 seconds
```

Ejemplo con datos inválidos (TC5):

```
Error: 'ABA' is not a valid number, skipping.
Error: '23,45' is not a valid number, skipping.
Error: '11;54' is not a valid number, skipping.
Error: 'll' is not a valid number, skipping.
COUNT: 311
MEAN: 241.49511400651465
MEDIAN: 241.0
MODE: 393.0
SD: 145.46484786056646
VARIANCE: 21229.17236166996
Elapsed Time: 0.001473 seconds
```

P2 — `convertNumbers.py`

Convierte números enteros a representación binaria y hexadecimal.

Requisito	Implementación
Req 1. Invocación por línea de comandos	`python convertNumbers.py archivo.txt`
Req 2. Conversión a binario y hexadecimal	Divisiones sucesivas, sin `bin()` / `hex()`
Req 3. Manejo de datos inválidos	Se reportan en consola y la ejecución continúa
Req 4. Nombre del programa	`convertNumbers.py`
Req 5. Formato de invocación	`python convertNumbers.py fileWithData.txt`
Req 6. Escala: cientos a miles	Probado con archivos de 200 elementos
Req 7. Tiempo de ejecución	Se muestra en pantalla y en `ConvertionResults.txt`
Req 8. PEP-8	pylint 10.00/10

Algoritmos utilizados:

- **Binario**: divisiones sucesivas entre 2, negativos en complemento a dos (10 bits)
- **Hexadecimal**: divisiones sucesivas entre 16, negativos en complemento a dos (40 bits)

Ejemplo de ejecución (TC3, con negativos):

```
ITEM     VALUE     BIN       HEX
1        -39       1111011001          FFFFFFFFD9
2        -36       1111011100          FFFFFFFFDC
3        8         1000      8
4        34        100010    22
...
Elapsed Time: 0.001480 seconds
```

Ejemplo con datos inválidos (TC4):

```
Error: 'ABC' is not a valid integer, skipping.
Error: 'ERR' is not a valid integer, skipping.
Error: 'VAL' is not a valid integer, skipping.
ITEM     VALUE    BIN      HEX
1        -39      1111011001         FFFFFFFFD9
...
Elapsed Time: 0.001122 seconds
```

P3 — `wordCount.py`

Cuenta la frecuencia de cada palabra distinta en un archivo.

Requisito	Implementación
Req 1. Invocación por línea de comandos	`python wordCount.py archivo.txt`
Req 2. Palabras distintas y frecuencias	Conteo con diccionario, sin `collections.Counter`
Req 3. Manejo de datos inválidos	Archivos vacíos o sin palabras se manejan correctamente
Req 4. Nombre del programa	`wordCount.py`
Req 5. Formato de invocación	`python wordCount.py fileWithData.txt`
Req 6. Escala: cientos a miles	Probado con archivos de hasta 5,000 palabras
Req 7. Tiempo de ejecución	Se muestra en pantalla y en `WordCountResults.txt`
Req 8. PEP-8	pylint 10.00/10

Ejemplo de ejecución (TC2):

```
amongst   4
brass     4
chain     4
doc       4
...
pre       3
wood      3
advantages          1
afternoon           1
...
Grand Total         184
Elapsed Time: 0.001244 seconds
```

Validación con pylint

Los tres programas obtuvieron la calificación máxima:

```
$ pylint P1/source/computeStatistics.py
Your code has been rated at 10.00/10

$ pylint P2/source/convertNumbers.py
Your code has been rated at 10.00/10

$ pylint P3/source/wordCount.py
Your code has been rated at 10.00/10
```

Se configuró `.pylintrc` con `module-naming-style=any` para permitir los nombres de archivo solicitados en la actividad (`computeStatistics.py` , `convertNumbers.py` , `wordCount.py`), como se indica en el punto 3 de las indicaciones adicionales.

Casos de prueba y resultados

P1 — Compute Statistics (7 TCs)

TC	Elementos	Datos inválidos	Resultado
TC1	400 números	No	Correcto
TC2	1,977 números	No	Correcto
TC3	12,624 números	No	Correcto
TC4	12,624 números (decimales)	No	Correcto
TC5	311 elementos	Sí (ABA, 23,45, 11;54, ll)	Correcto
TC6	3,000 números grandes	No	Correcto
TC7	12,769 elementos	Sí (ABBA, ERROR)	Correcto

P2 — Convert Numbers (4 TCs)

TC	Elementos	Rango	Datos inválidos	Resultado
TC1	200 enteros	Positivos grandes	No	Correcto
TC2	200 enteros	Positivos grandes	No	Correcto
TC3	200 enteros	-50 a 50	No	Correcto
TC4	200 elementos	-50 a 50	Sí (ABC, ERR, VAL)	Correcto

P3 — Word Count (5 TCs)

TC	Palabras	Resultado
TC1	100	Correcto
TC2	184	Correcto
TC3	500	Correcto
TC4	1,000	Correcto

TC	Palabras	Resultado
TC5	5,000	Correcto

La evidencia completa de cada ejecución se encuentra en:

- `P1/results/` — `execution_log.txt` + archivos `TC{n}_StatisticsResults.txt`
- `P2/results/` — `execution_log.txt` + archivos `TC{n}_ConvertionResults.txt`
- `P3/results/` — `execution_log.txt` + archivos `TC{n}_WordCountResults.txt`

Pruebas automatizadas

Se implementó una suite de pruebas con `pytest` que ejecuta cada programa contra los casos de prueba y compara los resultados automáticamente:

```
$ make test
======================== 20 passed in 1.22s ========================
```

Suite	Tests	Resultado
P1 — Compute Statistics	7 funcionales + 1 pylint	8/8 pass
P2 — Convert Numbers	5 funcionales + 1 pylint	6/6 pass
P3 — Word Count	5 funcionales + 1 pylint	6/6 pass
Total	**20**	**20/20 pass**

Estructura del repositorio

```
good-programming-practices/
├── P1/
│   ├── source/computeStatistics.py
│   ├── tests/TC1.txt … TC7.txt
│   └── results/                    ← Evidencia de ejecución
├── P2/
│   ├── source/convertNumbers.py
│   ├── tests/TC1.txt … TC4.txt
│   └── results/
├── P3/
│   ├── source/wordCount.py
│   ├── tests/TC1.txt … TC5.txt
│   └── results/
├── aux/                            ← Datos originales del profesor
├── tests/                          ← Suite de pruebas automatizadas (pytest)
├── .github/workflows/              ← CI/CD con GitHub Actions
├── .pylintrc
├── Makefile
└── README.md
```

Relación entre carpetas

Carpeta	Contenido
`aux/`	Archivos de entrada y resultados esperados proporcionados por el profesor. Son los datos de referencia y no se modifican.
`P{n}/tests/`	Copias de los archivos de entrada de `aux/`, organizadas por programa para ejecución independiente.
`P{n}/results/`	Salidas generadas al ejecutar cada programa con cada caso de prueba. Sirven como evidencia documentada de las corridas exitosas.
`P{n}/source/`	Código fuente del programa correspondiente.

Cómo ejecutar

```
# Ejecutar un programa individual
python P1/source/computeStatistics.py P1/tests/TC1.txt
python P2/source/convertNumbers.py P2/tests/TC1.txt
python P3/source/wordCount.py P3/tests/TC1.txt

# Ejecutar todas las pruebas
make test

# Ejecutar pylint en los 3 programas
make lint

# Ejecutar todo (lint + tests)
make all
```