Nota: 5 (cinco) - Corrigió: Lucho

# 107587 - P2

## Gallino, Pedro - Práctica Alan

| Ej. 1 | Ej. 2 | Ej. 3 |
|---|---|---|
| M | B | B= |
| N/E | Ok | Ok |

## » _mensaje_docente.txt

```
Se modificó el código del ejercicio 2 para que no reciba la ruta de los archivos como parámetros, de esta
forma pasa las pruebas.

El resultado de la ejecución se realiza en base a las pruebas que les damos. Entonces para futuros parcialitos
no modifiques las pruebas que tenés localmente.
```

## » ej2.py

```
# Se tiene un archivo CSV de tres columnas llamado `operaciones.csv`. Las columnas son:
#   - cuenta (cadena indicando el nombre de la cuenta)
#   - operacion (tipo de la operación: `extraccion` o `deposito`)
#   - monto (valor de la operación, un entero positivo).
# El archivo no tiene errores, está ordenado por el campo Cuenta y tiene encabezado con
# el nombre de cada columna.

# Se pide escribir una funcion `calcular_balances` que genere un archivo `balances.csv`
# con el balance de cada cuenta tras procesar las operaciones
# presentes en el archivo `operaciones.csv`.
# El archivo debe tener el formato y encabezado `cuenta,balance`.

# Notas:
# - Se debe asumir un balance inicial de 0.
# - Las extracciones restan al valor del balance y los depósitos suman al mismo,  y si se desea
# realizar una extraccion de mas plata que el monto disponible, la operacion no se realiza.
# - Al finalizar la ejecución de la función (haya ocurrido un error o no), todos los archivos
# abiertos deben quedar cerrados.


# Escribir debajo de este comentario el código del ejercicio

import csv

def calcular_balances():
    diccionario = {}
    with open('operaciones.csv') as original, open('balances.csv', "w") as destino:

        lector = csv.DictReader(original)
        for fila in lector:
            cuenta = fila["cuenta"]
            operacion = fila["operacion"]
            monto = fila["monto"]    <-- Es preferible hacer int(fila['monto']) acá
            if operacion == "deposito":
                diccionario[cuenta] = diccionario.get(cuenta, 0) + int(monto)
            elif operacion == "extraccion":
                if int(monto) > diccionario.get(cuenta, 0):
                    continue
                elif int(monto) <= diccionario.get(cuenta, 0):
                    diccionario[cuenta] = diccionario.get(cuenta, 0 ) - int(monto)

        destino.write(f"cuenta,balance\n")

        for clave in diccionario:                                   #considero que todos los balances
y los nombres de las cuentas entran en la memoria RAM
            destino.write(f"{clave},{diccionario[clave]}\n")
```

```
# Pruebas

calcular_balances()

with open('balances.csv') as f:
    lineas = f.read().splitlines()
    assert 'Cuenta de Nora,27' in lineas
```

## » ej3.py

```
# Se pide implementar una clase `Semaforo` con tres luces (roja, amarilla y verde)
# y el siguiente comportamiento:

# - Al instanciar la clase se debe ver la luz roja prendida.

# - Debe tener un metodo `siguiente` para apagar la luz actual y encender la siguiente
# (el orden de encendido de las luces es `roja->amarilla->verde->roja->amarilla->...`)

# - Debe tener un metodo `apagar` que apague el semaforo. En un semaforo apagado todas las
# luces estan apagadas y no se puede encender ninguna. Si se intenta usar el metodo
# `siguiente` mientras el semaforo esta apagado se debe lanzar una excepcion del tipo ValueError.

# - Debe tener un metodo `encender` que encienda el semaforo. Al encender el semaforo
# el estado de las luces debe ser el mismo que tenia antes de apagarlo.

# - Al imprimir una instancia de la clase semaforo, debe mostrar qué luz esta prendida.
# Si el semaforo esta apagado, debe mostrar que esta apagado.

# La representacion interna de la clase es a criterio pero debe ser acorde al comportamiento y no agregar
complejidad.


# Escribir debajo de este comentario el cdigo del ejercicio

class Semaforo:

    def __init__(self):

        self.prendido = True

        self.color = "roja"

        self.color_seguimiento = ["verde","amarilla","roja"]

    def __str__(self):

        return f"{self.color}"

    def siguiente(self):

        if self.prendido == False:
            raise ValueError("El semáforo está apagado")

        elif self.prendido == True:

            if self.color_seguimiento[-1] == "roja":
                self.color_seguimiento.append("amarilla")
                self.color_seguimiento.pop(0)
                self.color = "amarilla"

            elif self.color_seguimiento[-1] == "amarilla":
                self.color_seguimiento.append("verde")
                self.color_seguimiento.pop(0)
                self.color = "verde"
```

No hace falta

Esto resuelve (por el color "apagado", ver corrección mas abajo) pero no es correcto.

este elif no hace falta (hacer raise es similar a hacer return, no continua la ejecución hacia abajo)

Esto agrega una complejidad innecesaria

```
            elif self.color_seguimiento[-1] == "verde":
                self.color_seguimiento.append("roja")
                self.color_seguimiento.pop(0)
                self.color = "roja"

    def apagar(self):

        if self.prendido == False:
            raise ValueError("El semáforo ya está apagado")

        else:
            self.prendido = False
            self.color = "apagado"
    def encender(self):

        if self.prendido == True:
            raise ValueError("El semáforo ya está prendido")

        else:
            self.prendido = True
            self.color = self.color_seguimiento[-1]



# Pruebas

s = Semaforo()
assert 'roja' in str(s)
s.siguiente()
assert 'amarilla' in str(s)
s.siguiente()
assert 'verde' in str(s)
s.siguiente()
assert 'roja' in str(s)
s.apagar()
hay_excepcion = True
try:
    s.siguiente()
    hay_excepcion = False
except ValueError:
    pass
assert hay_excepcion, "excepcion no fue lanzada"
s.encender()
assert 'roja' in str(s)
s.siguiente()
assert 'amarilla' in str(s)
```

**if not self.prendido: ...**

**Esto está mal. "apagado" no es un color; y dado que el color es un atributo interno a la clase, no es necesario que cuando el semáforo está apagado el color cambie (ya que hacia afuera el semáforo se comporta como debe, es decir, como un semáforo apagado).**