Examen final 2020-09-25

95.14/75.40 - Algoritmos y Programación I - Curso Essaya

Objetivo

Implementar la clase `Editor`, que representa el estado de un editor de texto.

Se dispone de los siguientes archivos:

- `editor.py`: implementación (inicialmente vacía) de la clase `Editor`
- `pruebas.py`: Pruebas automáticas
- `texto.txt`: Un archivo de texto

El archivo `pruebas.py` efectúa una serie de pruebas para verificar el correcto funcionamiento de la clase `Editor`.

La idea es agregar en `editor.py` todo el código necesario para que las pruebas automáticas pasen.

Hay 5 ejercicios. Es condición necesaria (pero no suficiente) para aprobar el examen que haya 3 ejercicios OK.

Salida del programa

Al ejecutar el programa (`python3 pruebas.py`), se ejecutan todas las pruebas, y se imprime el resultado de cada ejercicio (OK o FAIL), junto con la cantidad de ejercicios OK. Ejemplo:

```
$ python3 pruebas.py
ejercicio_1: OK
ejercicio_2: OK
ejercicio_3: OK

Traceback (most recent call last):
File "pruebas.py", line 237, in main
    ejercicio()
File "pruebas.py", line 148, in ejercicio_4
    assert ok
AssertionError

ejercicio_4: FAIL
ejercicio_5: OK
Cantidad de ejercicios OK: 4
```

Recordar: es condición necesaria (pero no suficiente) para aprobar el examen que haya al menos 3 ejercicios OK.

La clase Editor

La clase `Editor` modela el estado interno de un editor de texto (pensar en cualquier editor de texto plano: Bloc de notas, Gedit, VSCode, Sublime, etc).

- El **editor** mantiene el contenido de un archivo de **texto**, que se va modificando a medida que el usuario hace acciones.
- El texto conceptualmente está formado por una secuencia de **líneas**.
- Una **línea** es básicamente una secuencia de caracteres que termina con el caracter `'\n'`, y no contiene ningún caracter `'\n'` en el medio.
- El editor siempre tiene **al menos una línea** de texto.
- Cada línea puede tener una longitud arbitrariamente grande, e independiente de las otras líneas. Cada línea tiene **al menos un caracter** (ya que el `'\n'` debe estar siempre como último caracter).
- El editor además maneja el estado del **cursor**, que indica la posición en la que el usuario insertará los caracteres al escribir.
- La posición del cursor está dada por un número de **línea** y un número de **columna**.
- Las líneas y columnas se cuentan a partir de 1. Es decir que si el texto tiene 10 líneas, se numeran de 1 a 10, y si una línea tiene 15 caracteres (incluyendo el `'\n'` final), las columnas en esa línea se numeran del 1 al 15, siendo la columna 15 la correspondiente al `'\n'`.

Descripción de las pruebas

`ejercicio_1`: Prueba el funcionamiento básico de la clase `Editor`. Sin esta prueba funcionando probablemente no se pueda pasar ninguna de las otras pruebas.

Métodos a implementar:

- `__init__`: Recibe una cadena de texto correspondiente al contenido completo del archivo. Crea una instancia del `Editor` con el contenido del archivo y el cursor en la posición `(1, 1)` (primera línea, primer caracter).

  Notar que la cadena recibida puede no cumplir con las restricciones que impone el editor. El editor se debe encargar de agregar lo necesario para cumplirlas:

  - La cadena puede estar vacía. El editor debe siempre tener al menos una línea de texto.
  - La última línea de la cadena puede no terminar con `'\n'`. Dicho caracter debe ser agregado en el editor.
- `cantidad_lineas`: Devuelve la cantidad de líneas que contiene el texto.
- `__str__`: Devuelve una cadena conteniendo el texto completo.
- `leer_archivo`: Recibe la ruta de un archivo de texto y devuelve una instancia de `Editor` con el contenido del archivo.

  Nota: esta es una función suelta, no es un método de `Editor`.
- `escribir_archivo`: Recibe la ruta de un archivo y escribe en el mismo el texto completo.

`ejercicio_2`: Prueba el movimiento del cursor, que ocurre cuando el usuario presiona las teclas de dirección del teclado.

Importante: en ningún caso el cursor puede quedar posicionado en un lugar inválido (línea o columna fuera de rango). Siempre que el usuario intente moverse fuera de los límites del texto, se hará alguna corrección para que el cursor quede en una posición válida.

Métodos a implementar:

- `cursor_pos`: Devuelve una tupla con el número de línea y columna del cursor.
- `cursor_caracter`: Devuelve el caracter sobre el cual el cursor está posicionado.
- `mover_derecha`: Mueve el cursor un caracter hacia la derecha.

  Si el cursor estaba en el último caracter de la línea, se moverá al primer caracter de la línea siguiente (en caso de ser posible).

- `mover_izquierda`: Mueve el cursor un caracter hacia la izquierda.

  Si el cursor estaba en el primer caracter de la línea, se moverá al último caracter de la línea anterior (en caso de ser posible).

- `mover_arriba`: Mueve el cursor un caracter hacia arriba.

  Si la línea de arriba es más corta que el número de columna actual, se posiciona el cursor en el último caracter.

- `mover_abajo`: Mueve el cursor un caracter hacia abajo.

  Si la línea de abajo es más corta que el número de columna actual, se posiciona el cursor en el último caracter.

- `mover_a`: Recibe un número de línea y columna y mueve el cursor a esa posición.

  Si la línea y/o columna están fuera de los límites, se ajustarán para que el cursor quede en una posición válida.

`ejercicio_3`: Prueba que el usuario pueda escribir texto y que el mismo se inserte en el lugar adecuado.

Nota: este ejercicio depende de que el cursor funcione, aunque solo es necesario hacer una implementación básica de la función `mover_a` del ejercicio 2.

Métodos a implementar:

- `insertar`: Recibe una cadena de texto que representa una secuencia de caracteres ingresados mediante el teclado. Por cada caracter, el mismo es insertado en la posición del cursor, y el cursor avanza una posición hacia la derecha.

  Algunos de los caracteres ingresados pueden ser `\n`, el cual luego de ser insertado divide la línea actual en dos líneas; todos los caracteres a la derecha del cursor pasan a formar una línea nueva abajo de la línea actual, y el cursor se posiciona al inicio de la misma.

`ejercicio_4`: Palabras más frecuentes

Métodos a implementar:

- `palabras_mas_frecuentes`: Devuelve una lista de tuplas `(palabra, cantidad)`, ordenada de mayor a menor por `cantidad`, indicando todas las palabras contenidas en el texto, y la cantidad de veces que aparece cada una.

  Una palabra se considera como cualquier secuencia de caracteres consecutivos que no incluyen el caracter `'\n'` o un espacio.

  Por ejemplo, en la cadena `'  hola,    que\n tal\n'`, las palabras son `'hola,'` (notar que la coma se considera parte de la palabra), `'que'`, `'tal'`.

`ejercicio_5`: Buscar

Métodos a implementar:

- `buscar`: Recibe una cadena que no incluye `'\n'`. Devuelve una lista de tuplas `(linea, columna)` con todas las posiciones en el texto en las que se encuentra la cadena.