PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN
IIC2233 - PROGRAMACIÓN AVANZADA

Actividad 07

2^o semestre 2017
21 de septiembre de 2017

Programación funcional: decoradores

Introducción

Luego de un increíble 18, vuelves con todas las ganas a tu curso favorito, ¡Programación Avanzada! Sin embargo, debido a ~~la PrograFonda~~ el gran descanso que tuviste, te das cuenta que te es difícil entender lo que dice el profesor, los ayudantes o incluso tus compañeros. Acudes, desesperado, a **Hernán el Iluminador**, el cual te regala su última creación: la Librería de Procesamiento Natural de Lenguaje (NLP por sus siglas en ingles). Esta increíble herramienta te permite reducir texto a sus partes más esenciales, acercándote a la idea de que, un día, tus programas puedan recibir órdenes en lenguaje natural, pero por ahora te ayudara a entender lo que te dicen.

Sin embargo, Hernán no tuvo el tiempo suficiente para implementar todas las funciones, así que hay ciertas funcionalidades (un registro de acciones, arreglo de argumentos, etc..) que no están implementadas. **No puedes cambiar el código original**, ya que cambiar código de Hernán sería un sacrilegio, pero si puedes usar tu conocimiento de decoradores para arreglar las funciones sin cambiar el código. Armado con esta herramienta, y una conveniente lista de decoradores a implementar (que puedes encontrar más abajo), sabes que puedes lograrlo.

Programa

Se deberán crear los siguientes decoradores:

- `verify_types(input_types, output_types)`: Verifica que el primer argumento de una función sea del tipo `input_type` y que el valor de retorno sea del tipo `output_type`. En caso contrario, lanzar una excepción de la clase `TypeError`.
- `to_lowercase`: Si el primer argumento de la función es un *string*, lo convierte a minúsculas. Si el primer argumento es una lista, debe convertir a minúsculas todos los *strings* que ésta contenga.
- `check_file`: Revisa que el primer argumento de la función corresponda al *path* de un archivo existente. Si archivo no existe, la función debe retornar `None`.

- `timer`: Imprime el tiempo, en milisegundos, que se demora en ejecutarse la función decorada.
- `remove_special_characters`: Si el primer argumento de la función decorada es un *string*, debe eliminar de él todos los caracteres que no sean alfanuméricos ni espacios. Si el primer argumento es una lista, debe hacer esto en todos los *strings* que contenga la lista.
- `get_stats`: Imprime el número de caracteres antes y después de aplicar la función decorada. Debes asumir que el primer argumento de la función será el texto original y el retorno de la función será el texto modificado, ambos pueden ser o una *string* o una lista de *strings*. En caso de que sea una lista de *strings*, el número de caracteres será la suma de los caracteres de todos los *strings*.
- `logging`: Cada vez que la función sea ejecutada, se debe guardar en un archivo llamado `log.txt` un registro con el nombre de la función, los argumentos que recibió y su resultado. Si `log.txt` ya existe, el registro debe agregarse al final del archivo.

Funciones a decorar

Es importante señalar que el uso de cualquiera de las funciones debe quedar registrado.

- `tokenize`: A esta función se le debe verificar el tipo de `input` y `output` y tomar el tiempo que demora en ejecutarse.
- `remove_stopwords`: A esta función se le debe tomar el tiempo que demora, se le deben entregar las letras en minúsculas, verificar el tipo de `input` y `output`, e imprimir la cantidad de letras del argumento y del resultado.
- `apply_stem`: Esta función debe imprimir la cantidad de letras que se le entregan como argumento, así como las del resultado.
- `save`: A esta función se le debe medir el tiempo que demora.
- `read`: A esta función se le debe medir el tiempo que demora y se debe comprobar que el argumento entregado exista.

Notas

- Puedes usar `time()` de la librería `time` para tomar el tiempo.
- Puedes usar `isfile()` de la librería `os` para ver si un archivo existe.
- Recuerda que puedes utilizar más de un decorador por función.
- **Sólo puedes modificar el código original con decoradores. Cualquier otro cambio en la estructura del sistema significará un 1.0 inmediato.**

Requerimientos

- (5.20 ptos) Decoradores funcionan correctamente.
  - (0.80 ptos) Decorador `verify_types` funciona correctamente.

    - (0.80 ptos) Decorador `to_lowercase` funciona correctamente.
    - (0.80 ptos) Decorador `get_stats` funciona correctamente.
    - (0.80 ptos) Decorador `timer` funciona correctamente.
    - (0.80 ptos) Decorador `remove_special` funciona correctamente.
    - (0.60 ptos) Decorador `check_file` funciona correctamente.
    - (0.60 ptos) Decorador `logging` funciona correctamente.
- (0.80 ptos) Decorar correctamente el programa.

Entrega

- **Lugar:** En su repositorio de Github en la **carpeta** `Actividades/AC07/`
- **Hora:** 16:55
- Si está trabajando en pareja, basta con que un miembro suba la actividad. Si se suben actividades distintas, se corregirá una de las dos al azar.