Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2233 Programación Avanzada (2020-2)

Actividad Bonus

Entrega

- **Fecha y hora:** miércoles 16 de diciembre de 2020, 20:00
- **Lugar:** Repositorio personal de GitHub — Carpeta: `Bonus`

Importante: Nota que la carpeta de entrega no existe actualmente en tu repositorio personal. Por eso debes crear la carpeta `Bonus`, al mismo nivel que las carpetas `Tareas` y `Actividades`, y hacer y entregar tu desarrollo en ella.

Objetivos

- Tener la oportunidad de obtener bonificación en calificaciones del curso.
- Utilizar y construir expresiones regulares para extraer información de textos, usando la biblioteca `pyrematch`

Introducción

En esta actividad analizaremos el contenido de las páginas de Wikipedia a través de las herramientas que nos da la biblioteca `REmatch`. Para esto, extraeremos el contenido de una página de Wikipedia, usando la biblioteca de Python `wikipedia`, la cual posee métodos que nos permiten acceder a páginas de Wikipedia a través de código en Python. Para instalar la biblioteca `wikipedia`, junto a la librería `pyrematch`, ejecuta:

```
pip3 install --upgrade wikipedia pyrematch
```

Y prueba que esté instalada correctamente con este código, que accede a la página de Wikipedia sobre Chile (en inglés).

```python
import wikipedia

# Solicitar el contenido de la página en Wikipedia para chile
texto = wikipedia.page("Chile", auto_suggest=False).content

# Imprimir los primeros 2000 caracteres
print(texto[:2000], '...')
```

Ahora la variable `texto` tiene el contenido de la página de Wikipedia correspondiente a Chile (versión inglés), en su versión fuente (*source*) incluyendo el texto de la página y los elementos del lenguaje que Wikipedia usa para organizar sus páginas en secciones, subsecciones, y subsubsecciones (incluyendo más

"subs" si es necesario). Esto significa que se han eliminado otros elementos que suelen desplegarse en el navegador, como marcadores HTML, *links*, e *infoboxes*, que no nos interesan en esta actividad.

Explora las primeras líneas del *string* `texto` y compáralas con el contenido de la página que puedes ver en tu navegador: https://en.wikipedia.org/wiki/Chile. Podrás ver que en el formato que has descargado, las secciones de la página se especifican escribiendo `== Nombre sección ==` en una sola línea del texto. Aquí es importante observar que entre el último `=` y el nombre de la sección hay un espacio, y similarmente al final del nombre hay un espacio antes del `=`. Será importante para definir nuestra expresiones regulares el que se especifique en una sola línea de texto, sin ningún caracter más. Similarmente, una subsección se especifica con `=== Nombre subsección ===`, y el delimitador para la subsubsección contiene cuatro símbolos `=`, esto es `==== Nombre subsubsección ====`.

Usando `REmatch`

En esta evaluación usarás la biblioteca `REmatch` de expresiones regulares para extraer contenido como secciones y subsecciones de una página de Wikipedia. Tendrás que implementar **seis consultas**. Para cada una de ellas será necesario definir una patrón (expresión regular), compilarlo en `REmatch` para obtener un objeto `regex`, e iterar sobre los resultados obtenidos usando `regex.findIter(texto)`. Para cada resultado deberás acceder al contenido y la posición de un *match*.

Figura 1: Logo de REmatch

A modo de ejemplo, el formato general de la respuesta para cada consulta debería ser de esta manera:

```python
import pyrematch as re

regex = re.compile(patron)

for match in regex.finditer(texto):
    # extraer los resultados
    pass

return #retornar los resultados solicitados
```

Consultas

Las consultas deben escribirse en el archivo entregado `consultas.py`. Los patrones a utilizar se encuentran definidos en variables de nombre `PATRONX`, donde `X` puede ser desde `1` hasta `6`. Dentro del mecanismo de corrección automática, cada consulta recibirá como argumento el `PATRONX` correspondiente. **No modifiques los nombres de estas variables, ni las definas dentro de cada consulta, de lo contrario tu entrega no podrá ser evaluada**.

Cada consulta recibe dos argumentos: el patrón a aplicar (`patron`), y el texto sobre el cual se aplicará (`texto`). Como resultado, debe retornar una lista de tuplas, donde cada tupla contiene tres elementos en el orden que se describe en cada consulta ~~contiene información de un *match* y está compuesta por tres elementos en el orden: texto del *match* solicitado, posición de inicio del *match* solicitado, y posición de término del *match* solicitado.~~

A continuación se describen las consultas que debes implementar.

Consulta 1

`def consulta1(texto, patron)`. Esta consulta busca todos los títulos de secciones del documento y la posición donde aparecen.

Cada tupla debe contener los elementos, nombre de la sección, posición de inicio del nombre de la sección, y posición de término del nombre de la sección.

Un resultado parcial de esta consulta sobre la página de Chile de Wikipedia es:

```
[
    ('External links', 78875, 78889),
    ('Further reading', 78851, 78866),
    ('References', 78796, 78806),
    ...
]
```

Consulta 2

`def consulta2(texto, patron)`. Esta consulta va un nivel más abajo, y busca todos los títulos de subsecciones del documento y las posiciones donde aparecen.

Cada tupla de contener los elementos, nombre de la subsección, posición de inicio del nombre de la subsección, y posición de término del nombre de la subsección.

Un resultado parcial de esta consulta sobre la página de Chile de Wikipedia es:

```
[
    ('Citations', 78832, 78841),
    ('Notes', 78816, 78821),
    ('Cultural heritage', 77603, 77620),
    ('Sports', 74389, 74395),
    ...
    ('Early history', 3876, 3889)
]
```

Consulta 3

`def consulta3(texto, patron)`. Esta consulta va aún más adentro de una subsección, y buscamos el contenido cada subsubsección del documento y sus posiciones.

Cada tupla debe contener los elementos, nombre de la subsubsección, posición de inicio del nombre de la subsubsección, y posición de término del nombre de la subsubsección.

Un resultado parcial de esta consulta sobre la página de Chile de Wikipedia es:

```
[
    ('Pinochet era (1973-1990)', 16830, 16854),
    ('Flora and fauna', 37068, 37083),
    ...
]
```

Consulta 4

`def consulta4(texto, patron)`. Esta consulta busca cada sección del documento que **NO tenga una subsección**, y las posiciones del **contenido** correspondiente.

Cada tupla debe contener los elementos: nombre de la sección, posición de inicio del **contenido de la subsección**, y posición de término del **contenido de la subsección**

El resultado de esta consulta sobre la página de Chile de Wikipedia es:

```
[
    ('Further reading', 78870, 78871)
    ('See also', 78714, 78792),
    ('Etymology', 2520, 3855),
    ...
]
```

Consulta 5

`def consulta5(texto, patron)`. Esta consulta es similar a `consulta3`, pero ahora buscamos cada subsección del documento y las posiciones no del nombre de la subsección, sino que de su contenido.

Cada tupla debe contener los elementos: nombre de la subsección, posición de inicio del **contenido de la subsección**, y posición de término del **contenido de la subsección**, incluyendo todas las subsecciones que pueda haber.

Un resultado parcial de esta consulta sobre la página de Chile de Wikipedia es:

```
[
    ('Early history', 3894, 4753),
    ('Spanish colonization', 4783, 7931),
    ('Independence and nation building', 7973, 11324),
    ...
    ('Citations', 78846, 78847)
]
```

Consulta 6

`def consulta6(texto, patron)`. En esta consulta buscamos cada subsección del documento que tenga **al menos una** subsubsección y extraer las posiciones de su contenido.

Cada tupla debe contener los elementos, nombre de la subsección, posición de inicio del contenido de la subsección, y posición de término del contenido de la subsección.

El resultado de esta consulta sobre la página de Chile de Wikipedia es:

```
[
    ('20th century', 11346, 19326),
    ('Biodiversity', 36534, 39652),
    ('Folklore', 73243, 73892)
]
```

Archivos entregados

Se entregan dos archivos:

- `consultas.py`. Este archivo contiene la definición de los patrones y de la consultas. **Debes completar este código, pero no debes cambiar los nombres de las variables que contienen los patrones, ni los nombres de las funciones que implementan las consultas**.
- `test.py`. Este archivo permite probar de manera automática el funcionamiento de tu código.

Probando la solución

Para probar el funcionamiento de tus consultas, se entrega el archivo `test.py`, que contiene una clase `Test`. Esta clase ya se encuentra implementada, y puedes modificarla como desees para efectuar tus propias pruebas. Si bien esta clase no es parte de lo que debes completar, esperamos que te ayude durante tu desarrollo para probar tu código y para encontrar posibles errores. Para efectuar la corrección utilizaremos un archivo construido de la misma manera.

La clase `Test` recibe una lista de argumentos, una lista de *outputs* (resultados) esperados y el nombre de la consulta a probar. Una vez instanciada la clase, el método `probar_casos` recorre la lista de argumentos, utiliza la consulta a evaluar para obtener su *output*, y lo compara con el *output* esperado. Si ocurre una excepción durante alguno de estos pasos, el resultado se considera incompleto o incorrecto. Solo si el resultado obtenido coincide con el esperado, el test se considerará correcto.

Corrección (automática) de evaluación

La corrección de esta evaluación será automática. Esto quiere decir que el puntaje asignado depende directamente de si las funcionalidades están implementadas correctamente o no, y no pasará por la corrección detallada de un ayudante. Para lograr esto, utilizaremos archivos de *test* que ejecutarán tu código, y a partir de este, obtener un *output* definido. Si este *output* coincide con el esperado, se considerará ese *test* como correcto.

Para cada función, se realizarán múltiples y variados *tests*, de distinta complejidad, que buscará detectar su correcta implementación. Cada *test* se considerará correcto solo si no ocurre una excepción durante la ejecución de la función y si el resultado coincide con el esperado. Si todos los *test* de una consulta son correctos, se asignará puntaje completo asignado a la consulta. De la misma forma, se asignará la mitad del puntaje si al menos el 75 % de los *test* son correctos. En caso contrario, no se asignará puntaje a la implementación de esa consulta.

A continuación se listan las funciones que serán corregidas, y los puntajes asociados a cumplir el 100 % de sus *tests* automáticos:

- Funciones en `consultas.py`.
  - (1.0 pt.) `consulta1`
  - (1.0 pt.) `consulta2`
  - (1.0 pt.) `consulta3`
  - (1.0 pt.) `consulta4`
  - (1.0 pt.) `consulta5`
  - (1.0 pt.) `consulta6`

Se puede acumular, en total, 6 puntos por una implementación completamente correcta. Esto se traduce en una bonificación de décimas sobre el promedio de una de las áreas del curso: Actividades o Tareas. Específicamente, se aplicará sobre el promedio que sin bonificación sea **más bajo** y la cantidad de décimas será proporcional a los puntos obtenidos bajo la siguiente fórmula:

$$n = \begin{cases} \mathbf{P} \times 0{,}5 & \text{si } \mathbf{T} \leq \mathbf{AC} \\ \mathbf{P} & \text{si } \mathbf{T} > \mathbf{AC} \end{cases}$$

Donde n es la cantidad de décimas, **P** es la cantidad de puntos obtenidos en la corrección automática, **T** es el promedio ponderado acumulado de Tareas y **AC** es el promedio de Actividades, calculado como descrito en el programa y ***syllabus***.

A modo de ejemplo, si los promedios de un estudiante en Tareas y Actividades son `3.90` y `4.75`, y obtiene `5.25` puntos en la corrección automática de esta evaluación, entonces sus promedios terminarían en `4.36` (`3.90 + 0.2625`) y `4.75`, respectivamente. Notar que tal estudiante pasaría a aprobar el curso tras este bonificación, ya que previamente no cumple con el requisito de promedio mínimo en Tareas.

Consideraciones importantes de entrega

En esta evaluación **no** se recibirán entregas atrasadas. Cualquier *commit* ***pusheado*** posterior a la hora de entrega no se considerará. Se aconseja, como siempre, **realizar *commits* y *pushs* parciales de sus avances**. Por ejemplo, al terminar de implementar una consulta, es un buen momento para subir un avance, de manera que no acumulen muchos cambios nuevos al hacer *push* cerca de la hora de entrega.

Para esta evaluación **no se evaluará el uso de `.gitignore`**. De todas formas, lo único necesario que debes entregar es el **módulo a completar**: `consultas.py`. No pasa nada si se sube `test.py`.

Tampoco habrá entrega de `README`, ya que la corrección será automática. Recuerda seguir la estructura de archivos entregada y **no alterar los nombres las variables `PATRONX` ni de las funciones a completar**, para asegurar estas puedan ser correctamente importadas.

Como mencionado al comienzo de este enunciado, se espera trabajes y entregues en tu repositorio personal, y en una carpeta **nueva** llamada `Bonus`. Se espera que usen los módulos base siguiendo la misma estructura que en el código entregado, al primer nivel de la carpeta `Bonus`:

```
Bonus
├── consultas.py
└── test.py
```

Restricciones y alcances

- Esta evaluación es **estrictamente individual**, y está regida por el **Código de honor de Ingeniería**.
- Tu programa debe ser desarrollado en Python 3.7.
- Tu entrega debe estar compuesta solo por el archivo `consultas.py` (de subir otros archivos estos serán ignorados).
- En esta actividad solo es necesario utilizar los módulos `pyrematch` y `wikipedia`, por lo que está prohibido utilizar módulos adicionales.

Las entregas que no cumplan con las restricciones del enunciado obtendrán la calificación mínima (0 puntos).