Teoría de Lenguajes

Curso 2023

Laboratorio 1 – Expresiones regulares

El objetivo de este laboratorio es el uso de expresiones regulares para dar soluciones a situaciones que se podrían encontrar en la práctica al trabajar con archivos o entradas de texto. Las expresiones regulares pasarán a ser una herramienta a tener en cuenta para utilizar en el tratamiento de textos, como ser archivos de texto en lenguaje natural, logs de programas, y archivos de código, entre muchos otros. Podrá utilizarlas de forma sencilla en lenguajes de programación (ej. pearl, python, ruby, etc.), a través de comandos de terminal (ej. grep, sed, awk, etc.) o editores de texto.

En este laboratorio se pide escribir un conjunto de rutinas en el lenguaje de programación Python 3. El trabajo está planteado con el fin de resolver los programas indicados con expresiones regulares, evitando utilizar estructuras de control (ej. for, while, If, etc.) u otras herramientas para resolverlos.

Modo de trabajo

- Se indican un conjunto de programas a realizar.
- Se imparte para cada programa archivos de entrada y archivos de salida que contienen la salida esperada para cada entrada.
- Cada grupo deberá implementar los programa pedidos considerando la especificación en esta letra, y las entradas, y sus correspondientes salidas.
- La salida del programa del estudiante deberá ser **exactamente igual** a la salida de referencia. Al igual que los nombres de los archivos de entrada, salida, y de los programas.
- Cada grupo debe entregar los programas codificados y el archivos integrantes.txt que se detalla más adelante en esta letra.

University Chatbot Dataset (kaggle)

Una aplicación popular en el área de procesamiento de lenguaje natural es la construcción de *chatbots o agentes conversacionales*. Un chatbot es un programa que interactúa en lenguaje natural con un usuario. Existen diversos mecanismos para la construcción de chatbots, entre los cuales se pueden destacar los basados en reglas o en aprendizaje automático. En general, es útil el uso de datasets para entrenar o evaluar un chatbot. En este laboratorio resolveremos un conjunto de problemas basados en el *University Chatbot Dataset*, el cual contiene una lista de intenciones con sus respectivas posibles entradas y respuestas para cada intención. El dataset completo se encuentra disponible en https://www.kaggle.com/datasets/niraliivaghani/chatbot-dataset

Sintaxis de archivos de entrada

Cada archivo JSON [5] contiene una lista de intenciones (campo *tag*), para las cuales se dan una lista de patrones de entrada (campo *patterns*) y una lista de posibles respuestas (campo *responses*). A continuación se muestra un ejemplo:

Ejemplo de archivo (0.json)

```
{
   "intents":[
 {
   "tag": "greeting",
   "patterns": [
      "Hi",
      "How are you?",
      "Is anyone there?",
      "Hello",
      "Good day",
      "What's up",
      "how are ya",
      "heyy",
      "whatsup",
      "??? ??? ??"
   ],
   "responses": [
      "Hello!",
      "Good to see you again!",
      "Hi there, how can I help?"
   ],
   "context_set": ""
 }
]
}
```

En cada programa se indicará una acción que debe realizar con el archivo de entrada, con un formato como el anterior, y la salida que debe desplegar. Cada grupo debe resolver todos los programas utilizando expresiones regulares tanto como le sea posible.

Programas a implementar

Implemente en Python 3 utilizando el módulo de expresiones regulares *re* los siguientes programas:

programa0.py

Despliega la primera intención (valor del campo *tag*) del archivo de entrada .

Con el archivo de ejemplo (0.json) se obtiene la siguiente salida:

```
greeting
```

programa1.py

Despliega todas las intenciones (valor del campo *tag*) del archivo de entrada.

Con el archivo de ejemplo (0.json) se obtiene la siguiente salida:

```
greeting
```

programa2.py

Despliega todas las intenciones (valor del campo *tag*) con terminación -ent, -ing, -or, u -on del archivo de entrada (ej. greeting).

Con el archivo de ejemplo (0.json) se obtiene la siguiente salida:

```
greeting
```

Nota: Las salidas de los programas 0, 1 y 2 coinciden para la entrada 0.json pero no necesariamente para otras.

programa3.py

Despliega las intenciones del archivo junto con la cantidad de *patterns* para cada intención.

Con el archivo de ejemplo (0.json) se obtiene la siguiente salida:

```
greeting 10
```

programa4.py

Despliega las intenciones del archivo junto con la cantidad de *patterns* y *responses* para cada intención.

Con el archivo de ejemplo (0.json) se obtiene la siguiente salida:

```
greeting 10 3
```

programa5.py

Despliega la cantidad total de intenciones, *patterns* y *responses* en el archivo.

Con el archivo de ejemplo (0.json) se obtiene la siguiente salida:

```
1 10 3
```

programa6.py

Simplifica los *patterns* y *responses* del archivo .json de entrada, dejando solamente, en caso de haber mas de uno, el primer *pattern* y la primera *response* para cada caso.

Con el archivo de ejemplo (0.json) se obtiene la siguiente salida:

```
{
   "intents":[
 {
   "tag": "greeting",
   "patterns": [
      "Hi"
   ],
   "responses": [
      "Hello!"
   ],
   "context_set": ""
 }
]
}
```

programa7.py

Simplifica el archivo .json de entrada sustituyendo todas las intenciones por T, los *patterns* por P y *responses* por R

Con el archivo de ejemplo (0.json) se obtiene la siguiente salida:

```
{
   "intents":[
 {
   "tag": "T",
   "patterns": [
      "P"
   ],
   "responses": [
      "R"
   ],
   "context_set": ""
 }
]
}
```

Herramientas a utilizar

Las rutinas deben estar codificadas en Python 3 [2], recomendamos utilizar Python 3.9 [1] o superior.

Para el módulo *re* de expresiones regulares de Python, sugerimos consultar los sitios [3] y [4].

lab1.zip

Junto con esta letra se imparte el archivo lab1.zip con el siguiente contenido:

- Los **archivos de entrada** (entradas/*.json).
- Los **archivos con las salidas esperadas de cada programa para cada entrada** (salidas_esperadas/*.txt).
- El directorio **programas**” con el **código** de **programa0.py**. El resto de los programas deben ser incluidos en este directorio.
- Un **script test.py** para ejecutar los programas y comparar las salidas.
- El programa ***diff.exe* para Windows** para comparar archivos.

Grupos

Los trabajos se deben realizar en grupos de hasta 4 estudiantes.

Entrega

La fecha límite para la entrega es el **21 de abril a las 23:59.**

Se habilitará un formulario en EVA para realizar la entrega.

Se debe entregar:

- Los archivos **programa{1,2,3,4,5,6,7}.py** con los programas implementados
- Un archivo **integrantes.txt** con las cédulas (sin puntos ni dígito de verificación) y nombre de los integrantes del grupo, uno por línea, separado por comas como se muestra en el ejemplo. La primera línea debe contener la cantidad de integrantes del grupo. Opcionalmente, se puede utilizar el final de este archivo para comentarios que el grupo considere pertinentes.

Ejemplo de archivo *integrantes.txt*:

```
3
7123456, ApellidoA1 ApellidoA2, NombreA1 NombreA2
8654321, ApellidoB1 ApellidoB2, NombreA1 NombreB2
9876543, ApellidoC1 ApellidoC2, NombreC1 NombreC2

Hicimos dos soluciones para el programa3, entregamos la que
consideramos mejor y dejamos comentada la otra.
```

Corrección

La corrección se realizará en el **sistema operativo Linux**. Recomendamos ejecutar los casos de prueba en Linux antes de entregar.

Si la ejecución termina abruptamente, o hay diferencia en los archivos de salida con la salida oficial, la solución no será considerada correcta.

Se deben respetar los nombres y forma de uso de los programas y el formato del archivo *integrantes.txt.* No se debe modificar el script test.py.

Referencias

[1] https://www.python.org/downloads/release/python-392/

[2] https://docs.python.org/3/

[3] https://docs.python.org/es/3/library/re.html

[4] https://www.w3schools.com/python/python_regex.asp

[5] https://www.json.org/json-es.html