Clase 6

Repaso

Expresiones Regulares

? El signo de interrogación tiene un uso extra además de hacer opcional un caracter. Cuando se usa justo después de un repetidor (es decir cuando se usa por ejemplo `.+?` ó `.*?`) su función se convierte en cambiar el comportamiento normal de la expresión regular, y lugar de expandirse tanto como pueda, se expandirá tan poco como pueda.

() Los paréntesis sirven para agrupar una expresión regular, de tal manera que los símbolos especiales que normalmente afectaban un solo caracter, ahora pueden afectar a un grupo.

[ ] Los corchetes también forman grupos de caracteres, pero en lugar de que el grupo se tome todo junto, la expresión va a coincidir con ALGUNO de los integrantes del grupo nada más.

[^] Además de eso, si el primer caracter dentro de los corchetes es ^ el comportamiento de la expresión regular es coincidir con todo lo que NO sea nada del grupo que está entre esos corchetes (sin contar el acento, obviamente).

[-] Por último, si se utilizan el símbolo de menos se pueden hacer rangos de valores dentro de los corchetes (para no escribirlos todos) por ejemplo `[a-z]` son todas las letras de la 'a' a la 'z' y `[0-9]` es el grupo de los dígitos.

Repaso

Expresiones Regulares

\d	Las expresiones regulares también tienen conjuntos predefinidos. La diagonal invertida con 'd' es el grupo de los dígitos.
\s	La diagonal invertida con 's' es el grupo de los espacios (espacio, tabuladore, salto de línea, retorno de carro).
\w	La diagonal invertida con 'w' son los caracteres alfanuméricos (letras y números) y el guión bajo (_).
\D\S\W	La diagonal invertida con letras mayúsculas coinciden con lo OPUESTO a sus contrapartes minúsculas.

> Las pleca se utiliza para obtener solo uno de un grupo de valores. Este símbolo se usa para separar expresiones regulares, pero NO se utiliza para un solo caracter, es decir, reconoce la expresión regular completa que le precede o sucede, no necesita paréntesis para eso. Por otro lado, lo más común es encontrarla DENTRO de los paréntesis.

```
meses=r"(Enero|Febrero|Marzo|Abril|Mayo|
        Junio|Julio|Agosto|Septiembre|
        Octubre|Noviembre|Diciembre)"
```

Ejercicio 13

Ahora le toca el turno a las fechas.

- Extraigan con expresiones regulares las fechas mencionadas en su texto.

Para aquellos que ponen mucha atención en sus documentos:

- Extraigan TODAS las fechas, incluso las que están escritas con letras.

Expresiones regulares

Por último hablemos de grupos y referencias.

- Los paréntesis son capaces de agrupar expresiones para usar instrucciones sobre más de una letra, cierto. Pero también se acuerdan del grupo que formaron, los cuáles se van enumerando.
- El primer paréntesis que se usa, forma al grupo \1, el segundo forma al grupo \2 y así sucesivamente.

```
expresion_regular=re.compile(r"\(([A-Z]\w*)\).*\1")
resultados_busqueda=expresion_regular.finditer(texto)

for resultado in resultados_busqueda:
        print(resultado.group(0))
        print("\n",resultado.group(1),"\n")
```

Expresiones regulares

- Las expresiones regulares de python (el módulo **re**) cuentan con funciones bastante útiles además de la búsqueda.
- Una de ellas es la sustitución. Para lo que se usa la función: **sub()**

```
expresion_regular=re.compile(r"([^\w\s][^A-Z\d])")
texto_nuevo=expresion_regular.sub(r" \1",texto)

print(texto_nuevo)
```

Ejercicio 14

El último programa separa de las palabras los signos de puntuación que ocurren de su lado derecho.

- Usen el texto de salida anterior y un nuevo paso de proceso para separar los signos de puntuación izquierdos, de modo que queden las palabras y los signos completamente separados por espacios.

Solución

- Esta vez también les presentaré aquí la solución del ejercicio.

```
expresion_derecha=re.compile(r"([^\w\s][^\A-Z\d])")
texto_nuevo=expresion_derecha.sub(r" \1",texto)
expresion_izquierda=re.compile(r"([^\A-Z\d][^\w\s])")
texto_ultimo=expresion_izquierda.sub(r"\1 ",texto_nuevo)

print(texto_ultimo)
```

- Ya que usaremos ese nuevo texto con todo bien separado.

Expresiones Regulares

- ¿Recuerdan nuestro tokenizador básico?
- Tenía algunas fallas, pero ahora que las cosas están bien separadas por espacios, podemos lograr algo mejor.

```
expresion_derecha=re.compile(r"([^\w\s][^\A-Z\d])")
texto_nuevo=expresion_derecha.sub(r" \1",texto)
expresion_izquierda=re.compile(r"([^\A-Z\d][^\w\s])")
texto_ultimo=expresion_izquierda.sub(r"\1 ",texto_nuevo)

tokens_lista=texto_ultimo.split()

for token in tokens_lista:
        print(token)
```

Expresiones Regulares

- Esta nueva versión de tokenizador funciona mejor gracias a las expresiones regulares.
- Aún le faltan cosas para ser del todo confiable. Pero por ahora podemos ver cómo se van formando muchas de las reglas que componen un tokenizador completo, y además es importante conocer las expresiones regulares como herramienta.
- Por supuesto, los tokenizadores, al igual que otras herramientas del PLN, están bastante estudiados y los hay a nuestra disposición sin tener que armar uno nosotros mismos.