Clase 4

import Python tiene un sinnúmero de funciones. Es necesario catalogarlas, de manera que no se lean todas siempre, eso sería sumamente pesado y sin ningún beneficio. Además de las funciones también tienen diferentes orígenes. Es por eso que Python tiene una instrucción para cuando buscan usar comandos que no se cargan por defecto. Esa instrucción es `import` seguida del nombre del módulo que van a importar.

Repaso

Lista de archivos

os El módulo `os` contiene instrucciones que le permiten a Python interactuar con el sistema operativo.

listdir() Dentro de éste módulo tenemos la función `listdir()`, pertenece al módulo `os` y por lo tanto se usa el punto para usarla. Esta función recibe como parámetro el nombre de una carpeta, y *regresa* una lista con los nombres de los archivos que contiene dicha carpeta.

Repaso

Funciones

sort() Una función de las listas. Permite ordenar los elementos que contiene la lista, tienen la opción de utilizar el parámetro: **`reverse=False`** si quieren que el orden sea descendente.

sorted() Esta NO es una función de las listas, es independiente, y recibe como parámetro a la lista. La diferencia radica en que esta *regresa* una NUEVA lista ordenada, una copia. La función anterior, MODIFICA la lista al ordenarla. Esta función también acepta el parámetro **`reverse=False`**.

Repaso

Funciones

split() Una función de las variables tipo texto. Permite segmentar el texto y coloca las partes en una lista. Como divisor utiliza los espacios, o bien, pueden ingresar un texto como parámetro en esta función, y lo usara como indicador de corte EN LUGAR del espacio (que es el divisor por defecto).

replace() Otra función de los textos. Permite reemplazar texto, recibe dos parámetros. El primero, es el texto que quieren reemplazar (bucar...). El segundo, es el texto con el que lo quieren reemplazar (reemplazar con...). Por defecto reemplaza TODAS las apariciones que encuentra, pero tiene un tercer parámetro opcional, un número para indicar cuántas apariciones del texto quieren reemplazar.

Ejercicio 6

Obtengan, en una lista, el vocabulario de su corpus completo.

- Utilicen la última versión de tokenizador que vimos, esta bien si incluyen símbolos en la lista, pero no deben estar pegados a las palabras.
- Ningún elemento debe estar repetido.

Yo se que quieren también algo más difícil:

- Obtengan por separado, el vocabulario de las RESOLUCIONES, de los ACUERDOS y de otros documentos diferentes.
- Además, para cada caso, SEPAREN los símbolos de las palabras, pero NO los pierdan.

Expresiones Regulares

Expresiones regulares

Hemos hablado sobre los problemas de reconocer cierto tipo de palabras o tokens. Sin embargo, son cosas que presentan ciertos patrones.

- Las siglas o abreviaturas, como: S.A. de C.V. o C.P.
- Precios y números, como: $16.99 , 3.1416 $25.
- Fechas, como: 15/Nov/1997 , 18/09/93.
- Correos electrónicos: gil@iingen.unam.mx

Hay muchas expresiones que siguen ciertos patrones, y es precisamente para detectar estas expresiones que nos pueden servir las expresiones regulares.

Expresiones regulares

Una expresión regular es una fórmula en un lenguaje especial que se utiliza para especificar clases particulares de texto que siguen algún patrón.

Expresiones regulares

- Python soporta el uso de expresiones regulares, con ayuda de un módulo llamado **`re`**

```
import re
```

- Comenzaremos con las expresiones regulares más simples, que son aquellas que coinciden consigo misma, esto es equivalente a buscar un texto concreto.

Expresiones regulares

```
import re

carpeta_nombre="Documentos\\"
archivo_nombre="P_IFT_290216_73_Acc.txt"

with open(carpeta_nombre+archivo_nombre,"r") as archivo:
        texto=archivo.read()

expresion_regular=re.compile(r"México")
resultado_busqueda=expresion_regular.search(texto)

print(resultado_busqueda.group(0))
```

- En este ejemplo utilizamos la función **compile()** del módulo **re** para definir una expresión regular, en este caso sólo usamos una palabra tal cual.

Expresiones regulares

- Este es el mismo ejemplo anterior, recortado para enfocarnos en la parte más importante.

```
expresion_regular=re.compile(r"México")
resultado_busqueda=expresion_regular.search(texto)

print(resultado_busqueda.group(0))
```

- El resultado de la función **`compile()`** lo almacenamos en la variable **`expresion_regular`**.
- La expresiones regulares (como la que ahora se encuentra dentro de la variable **`expresion_regular`**) cuentan con funciones propias. Aquí estamos usando la función **`search()`**, que sirve para encontrar la expresión regular en un texto dado.
- El resultado de la búsqueda se almacena en la variable **`resultado_busqueda`** para mostrarlo en pantalla más adelante.

Expresiones regulares

Es importante notar dos detalles en este proceso:

- Hay una '**r**' justo antes del texto, dentro de la definición de la expresión regular. Esto es para aclarar que se usará una expresión regular y que los símbolos especiales de Python no interfieran.
- Para mostrar el resultado, se usa: **group(0)**. Esto quiere decir que queremos el primer grupo (recuerden que los índices comienzan en 0) del resultado. Es posible hacer agrupaciones dentro de las expresiones regulares, lo veremos más adelante, por ahora, usen así la función para obtener el texto encontrado.

```
expresion_regular=re.compile(r"México")
resultado_busqueda=expresion_regular.search(texto)

print(resultado_busqueda.group(0))
```

Expresiones regulares

- Algo que tienen que considerar, es que la función **`search()`** va a regresar únicamente la primer coincidencia que encuentre, para encontrar todas, pueden usar la función **`finditer()`**

```
expresion_regular=re.compile(r"México")
resultados_busqueda=expresion_regular.finditer(texto)

for resultado in resultados_busqueda:
        print(resultado.group(0))
```

- Esta función tiene el mismo comportamiento que **`search()`** pero regresa todas las coincidencias, para entrar a cada una de ellas, es necesario usar un bucle para recorrer la variable como si fuera una lista. A esto se le llama un objeto iterable (sí, las listas son iterables).

Expresiones regulares

- Comencemos ahora con los símbolos especiales de las expresiones regulares.
- Veamos el punto '.'. El punto coincide con un caracter, el que sea.

```
expresion_regular=re.compile(r".")
resultado_busqueda=expresion_regular.search(texto)

print(resultado_busqueda.group(0))
```

```
expresion_regular=re.compile(r".")
resultados_busqueda=expresion_regular.finditer(texto)

for resultado in resultados_busqueda:
        print(resultado.group(0))
```

Expresiones regulares

- Sigamos ahora con el asterisco '*'.
- El asterisco permite cualquier cantidad de repeticiones del símbolo que lo precede.

```
expresion_regular=re.compile(r".*")
resultado_busqueda=expresion_regular.search(texto)

print(resultado_busqueda.group(0))
```

```
expresion_regular=re.compile(r".*")
resultados_busqueda=expresion_regular.finditer(texto)

for resultado in resultados_busqueda:
        print(resultado.group(0))
```

Expresiones regulares

- Acabamos de ver como funciona el asterisco junto con el punto, pero también funciona con letras, aquí les presento un ejemplo un tanto más confuso en su salida.

```
expresion_regular=re.compile(r"I*")
resultado_busqueda=expresion_regular.search(texto)

print(resultado_busqueda.group(0))
```

```
expresion_regular=re.compile(r"I*")
resultados_busqueda=expresion_regular.finditer(texto)

for resultado in resultados_busqueda:
        print(resultado.group(0))
```

Expresiones regulares

- Un efecto similar al asterisco lo logra el símbolo de suma '+'.
- La diferencia radica en que la suma significa una o más repeticiones, no cero o más.

```
expresion_regular=re.compile(r"I+")
resultado_busqueda=expresion_regular.search(texto)

print(resultado_busqueda.group(0))
```

```
expresion_regular=re.compile(r"I+")
resultados_busqueda=expresion_regular.finditer(texto)

for resultado in resultados_busqueda:
        print(resultado.group(0))
```

Expresiones regulares

- Pasemos al siguiente símbolo especial: '^'.
- Se utiliza para indicar el inicio del texto.

```
expresion_regular=re.compile(r"^.")
resultado_busqueda=expresion_regular.search(texto)

print(resultado_busqueda.group(0))
```

```
expresion_regular=re.compile(r"^.")
resultados_busqueda=expresion_regular.finditer(texto)

for resultado in resultados_busqueda:
        print(resultado.group(0))
```

Ejercicio 7

- Lean un archivo línea por línea.
- Para cada línea, busquen la expresión regular de la diapositiva anterior y muestren el resultado.

Expresiones regulares

- El siguiente símbolo especial: '$'.
- Se utiliza para indicar el final del texto.

```
expresion_regular=re.compile(r".$")
resultado_busqueda=expresion_regular.search(texto)

print(resultado_busqueda.group(0))
```

```
expresion_regular=re.compile(r".$")
resultados_busqueda=expresion_regular.finditer(texto)

for resultado in resultados_busqueda:
        print(resultado.group(0))
```

Ejercicio 8

Éste es muy fácil ya.

- Hagan lo mismo que en el ejercicio pasado, pero para la expresión regular de la diapositiva anterior.

Expresiones regulares

- El siguiente símbolo especial: '**?**'.
- Es parecido al símbolo '**+**'. Pero en lugar de ser 1 o más, es 1 o 0. Acepta que aparezca o la letra anterior.
- Esta vez solo usaremos el ejemplo iterativo, ya sabemos que el otro muestra el primer resultado del iterativo.

```
expresion_regular=re.compile(r"artículos?")
resultados_busqueda=expresion_regular.finditer(texto)

for resultado in resultados_busqueda:
        print(resultado.group(0))
```