* La librería re.

Es el módulo que contiene funciones y clases relativas a búsqueda de patrones mediante expresiones regulares.

* + In [1]:

import re

La función re.search().

* + Busca un patrón desde el principio de la cadena de caracteres. Al encontrar la primera conicidencia, regresa un objeto.

Sintaxis:

re.search(<patrón>, <cadena de caracteres>, <parámetros>)

Ej:

texto = "Azucar azu azucarado zucarita, asucar azurca azucar"

patron = "azu"

busqueda = re.search(patron, texto)

busqueda.string

'Azucar azu azucarado zucarita, asucar azurca azucar'

busqueda.span()

(7, 10)

busqueda.start()

7

busqueda.end()

10

re.search(patron, texto, re.I)

<\_sre.SRE\_Match object; span=(0, 3), match='Azu'>

La función re.split().

* + Busca todas las coincidencias de un patrón desde el principio de la cadena de caracteres y separa los elementos utilizando al patrón como separador. Regresa una lista de cadenas de caracteres con los textos separados. En caso de no encontrar coincidencias, regresa un objeto de tipo list con el texto original.

Sintaxis:

re.split(<patrón>, <cadena de caracteres>, <parámetros>)

Ej:

patron = ('azu')

texto = "Azucar azu azucarado zucarita, asucar azurca azucar"

re.split(patron, texto)

['Azucar ', ' ', 'carado zucarita, asucar ', 'rca ', 'car']

La función re.sub().

* + Busca todas las coincidencias de un patrón desde el principio de la cadena de caracteres y dichas coincidencias serán sustituida con un nuevo texto. Regresa una cadena de caracteres con el texto modificado.

Sintaxis:

re.sub(<patrón>, <texto a sustituir>, <texto>, <parámetros>)

Ej:

patron = 'azucar'

texto = "Azucar azu azucarado zucarita, asucar azurca azucar"

re.sub(patron, 'dulce', texto)

'Azucar azu dulceado zucarita, asucar azurca dulce'

La función re.compile().

* + Crea un objeto que incluye los siguientes métodos, los cuales se comportan de forma idéntica a las funciones del módulo re utilizando el patron que se ingresa como parámetro:

search().

findall().

finditer().

split().

sub().

subn().

match().

En cada módulo se ingresa un texto como parámetro para que el patrón sea aplicado en éste.

Sintaxis:

re.compile(<patrón>, <parámetros>)

patron = re.compile('el patron', re.I | re.MULTILINE

Algunos patrones con expresiones regulares.

* + Es posible construir patrones que no necesariamente corresponden a una cadena de caracteres única mediante las expresiones regulares.
    - El metacaracter "."

Indica que debe de haber un caracter en lugar del punto.

Ej:

texto = "Carro carcar carrreta caar craacar Carranza"

patron = "c.ar"

re.findall(patron, texto, re.I)

['caar']

patron = "ca..r"

re.findall(patron, texto, re.I)

['carrr']

patron = "..ar"

re.findall(patron, texto, re.I)

[' car', ' car', 'caar', 'acar', ' Car']

* + - El metacaracter "^".

Indica que el principio de la cadena de caracteres debe de coincidir con la expresión.

Ej:

texto = "Carro carcar carrreta caar craacar Carranza"

patron = "^car"

re.findall(patron, texto, re.I)

['Car']

texto = "Cuenta cuentos cuenta cuentas cuantas cuentas cuenta"

patron = "^.ue"

re.findall(patron, texto, re.I)

['Cue']

* + - El metacaracter "$".

Indica que el final de la cadena de caracteres debe de coincidir con la expresión.

Ej:

texto = "Carro carcar carrreta caar craacar Carranza"

patron = "rranza$"

re.findall(patron, texto, re.I)

['rranza']

* + - El metacaracter "\*".

Indica que puede haber 0 o más caracteres en la posición indicada.

Ej:

texto = "Carro carcar carrreta caar craacar Carranza"

patron = "ca\*r"

re.findall(patron, texto, re.I)

['Car', 'car', 'car', 'car', 'caar', 'cr', 'car', 'Car']

texto = "Carro carcar carrreta caar craacar Carranza"

patron = "ca\*"

re.findall(patron, texto, re.I)

['Ca', 'ca', 'ca', 'ca', 'caa', 'c', 'ca', 'Ca']

* + - El metacaracter "+".

Indica que puede haber 1 o más caracteres en la posición indicada.

Ej:

texto = "Carro carcar carrreta caar craacar Carranza caaar"

patron = "ca+"

re.findall(patron, texto, re.I)

['Ca', 'ca', 'ca', 'ca', 'caa', 'ca', 'Ca', 'caaa']

* + - El metacarcter "?".

Indica que pueden haber cero o una coincidencia con el caracter de la izquierda.

Ej:

texto = "Carro carcar carrreta caar craacar Carranza, caaar"

patron = "ca?r"

re.findall(patron, texto, re.I)

['Car', 'car', 'car', 'car', 'cr', 'car', 'Car']

* + - El uso de corchetes "[...]"

Indica un conjunto de caracteres.

Ej:

texto = "Azucar azu azecarado zucarita, asucar azurca azicar"

patron = "az[aeiou]car"

re.findall(patron, texto, re.I)

['Azucar', 'azecar', 'azicar']

* + - El uso de llaves "{m\_, \_n}"

Indica un rango de coincidencias del caracter.

Ej:

texto = "1211111111 11111 1111 11111 11 11 13"

patron = "1{3,4}"

re.findall(patron, texto, re.I)

['1111', '1111', '1111', '1111', '1111']

* + - Otros

Si sabemos que obtendremos más de una coincidencia, podemos usar el método findall, que recorre la cadena y devuelve una lista de coincidencias:

patron.findall('a455 a333b435')

['a455', 'a333']

O el método finditer, que devuelve un iterador que podemos usar en el bucle for:

for m in patron.finditer('a455 a333b435'):

print m.groups()

('a', '455') ('a', '333') ('b', '435')

Notas:

* Expresiones Regulares

Las expresiones regulares son parte de los lenguajes formales y corresponden a una secuencia de caracteres que definen un patrón.

Mediante el uso de lenguajes formales, es posible buscar patrones dentro de un flujo de texto.

* + Metacaracteres.

Python reconoce a los siguientes caracteres como componentes sintácticos de una expresión regular.

. ^ $ \* + ? { } [ ] \ | ( )

El uso de los metacaracteres se estudiará más adelante.