19 de mayo de 2024

INGENIERIA EN DESARROLLO DE SOFTWARE

Materia: FUNDAMENTOS DE PROGRAMACION II

1.3 Actividad. Manejo de cadenas y expresiones regulares.

Elaborado por: Jesús Angel Hernández Martínez

Contenido

Regex

Regex (Regular expressions), expresiones regulares son secuencias de caracteres que definen una serie de patrones utilizadas como instrucciones para ejecutarlas sobre un texto de entrada y producir una versión modificada de este.

Los componentes de la sintaxis de las regex son:

- Literales: Cualquier carácter se encuentra a sí mismo, a menos que se trate de un metacaracter con significado especial.
- Secuencias de escape: finales de línea, tabs, barras diagonales, etc.
- Clases de caracteres: se pueden especificar clases de caracteres encerrando una lista entre corchetes.
- Metacaracteres: son caracteres con significado especial.

Tipo	Patrón	Significado
Metacaracteres delimitadores	.	Cualquier carácter excepto el que representa una nueva línea
Secuencia de escape	\n	Nueva línea
Patrones básicos	\r	Retorno de carro
Secuencia de escape	\t	Tabulador horizontal
Secuencia de escape	\v	Tabulador vertical
Clases predefinidas	\w	Cualquier carácter que represente un numero o letra en minúscula
Clases predefinidas	\W	Cualquier carácter que represente un numero o letra en mayúscula
Clases predefinidas	\s	Espacio en blanco (nueva línea, retorno de carro, espacio y cualquier tipo de tabulador)
Clases predefinidas	\S	Cualquier carácter que no es un espacio en blanco
Clases predefinidas	\d	Numero entre 0 y 9 inclusive
Clases predefinidas	\D	Cualquier carácter que no es un numero
Metacaracteres delimitadores	$	Fin de cadena
Metacaracter de memorias	\	Escape para caracteres especiales
Secuencia de escape	\\	Barra diagonal inversa
Patrones básicos	[]	Rango, cualquier carácter que se encuentre entre corchetes
Patrones básicos	^[]	Cualquier carácter que no se encuentre corchetes.

Metacaracteres delimitadores	\b	Separación entre un numero y/o letra.
Metacaracteres de repetición(iteradores)	+	Una o más veces
Metacaracteres de repetición(iteradores)	*	Cero o más veces
Metacaracteres de repetición(iteradores)	?	Cero o una vez
Metacaracteres de repetición(iteradores)	{n}	El carácter se repite n veces
Metacaracteres de repetición(iteradores)	{n,}	Por lo menos n veces
Metacaracteres de repetición(iteradores)	{n,m}	Por lo menos n pero no más de m
Metacaracteres de repetición(iteradores)	*?	Cero o mas
Metacaracteres de repetición(iteradores)	+?	Una o mas
Metacaracteres de repetición(iteradores)	??	Cero o una
Metacaracteres de repetición(iteradores)	{n}?	Exactamente n veces
Metacaracteres de repetición(iteradores)	{n,?}	Por lo menos n veces
Metacaracteres de repetición(iteradores)	{n,m}?	Por lo menos n veces no mas de m
Metacaracteres delimitadores	^	Inicio de línea
Metacaracteres delimitadores	\A	Inicio de texto
Metacaracteres delimitadores	\Z	Fin de texto
Metacaracteres delimitadores	\B	Encuentra distinto límite de palabra
Secuencia de escape	\ooo	Carácter ASCII en notación octal
Secuencia de escape	\xhh	Carácter ASCII en notación hexadecimal
Secuencia de escape	\xhhhh	Carácter Unicode en notación hexadecimal
Metacaracter de alternativas	|	Separa alternativas de búsqueda
Metacaracter de subexpresiones	(...)	Es empleada para definir subexpresiones de expresiones regulares

Modulo *re*

Python incluye el módulo ***re*** para trabajar con regex, este módulo contiene funciones que nos permite realizar **búsquedas, sustituciones y modificadores.**

Para utilizar las funciones mencionadas podemos hacer uso de la compilación de la expresión regular o bien utilizarlas a nivel de modulo, ejemplo:

1- Compilar la expresión.
patron = re.compile(r"\bfoo\b") #significa busca la palabra foo
patron.search("la palabra foo está aquí")

Compilar la expresión es muy útil si vamos a buscar el mismo patron utilizando distintas funciones en distintas cadenas o bien si la búsqueda de este patron será recurrente a lo largo del desarrollo de un programa o en un bucle.

2- Expresión no compilada
re.search(r"\bfoo\b", "la palabra foo está aquí")

Función	Operación	Significado	Sintaxis
match()	Búsqueda	Determina si la regex tiene coincidencias solamente en el comienzo del texto.	re.match(patron, cadena)
search()	Búsqueda	Busca en todo el texto cualquier ubicación donde encuentre la coincidencia.	re.search(patron, cadena)
findall()	Búsqueda	Encuentra todos los subtextos donde haya una coincidencia y devuelve estas coincidencias como una lista.	re.findall(patron, cadena)
fullmatch()	Búsqueda	Retorna un objeto si y solo si la cadena completa coincide con el patron.	re.fullmatch(patron, cadena)
sub()	Sustitución	Encuentra los subtextos que coincidan con el patrón y los reemplaza con un nuevo texto, esta función recibe cuatro argumentos: 1- Expresión regular que indica el patron que será buscado 2- Cadena de texto que se usara como reemplazo 3- Cadena de texto donde se llevará a cabo el reemplazo 4- Opcional, numero de ocurrencias que deben ser reemplazadas	re.fullmatch(patrón, cadena reemplazo, cadena donde se aplicara el reemplazo, opcional*) *ocurrencias
split()	Separación	Divide el texto en partes, en cada lugar donde encuentre la coincidencia, el resultado es una lista con cada elemento(parte).	re.split(patrón, cadena, maximo_separaciones,opcional*) *banderas de compilacion

Ejemplos de Funciones

match()

match solo busca coincidencias al inicio del texto, es por eso que retorna None en la primera prueba.

```
>>> #EJEMPLOS
>>> #match
>>> multilinea = """2024 Mexicanos
... al grito de guerra
... y retiemble"
... """
>>> print(re.match(r"al",multilinea))
None
>>> print(re.match(r"2024",multilinea))
<re.Match object; span=(0, 4), match='2024'>
>>>
```

search()

La Función **search()** tiene como objetivo buscar una cadena dentro de otra, y recibe 2 argumentos:

1. Una expresión regular(regex) representada por un patrón.
2. Una cadena donde realizar la búsqueda.

Encontrar una cadena en otra.

```
In [1]: import re
        re.search(r"o","hola")

Out[1]: <re.Match object; span=(1, 2), match='o'>
```

```
In [2]: #re.search es un objeto del modulo re, al almacenar el objeto en una variable
        #podemos acceder a sus metodos, en este caso el metodo group()
        #que devolvera la subcadena.
        subcadena =  re.search(r"urban","suburban");
        print(subcadena.group());#

        urban
```

Encontrar una subcadena que contenga tres números seguidos.

```
>>> import re
>>> texto = "holamundo958holamexico123"
>>> re.search(r"\d\d\d",texto)
<re.Match object; span=(9, 12), match='958'>
>>> subtexto = re.search(r"\d\d\d",texto)
>>> print(subtexto.group())
958
>>>
```

findall()

Retorna una lista con todos los valores de coincidencia

```
>>> #findall()
>>> cadena = "345abcd78ghx678"
>>> print(re.findall(r'\d{3}',cadena)) #'\d{3}' significa numeros de 0-9 que se repitan 3 veces(grupos de 3 numeros cons
ecutivos)
['345', '678']
>>>
```

fullmatch()

Retorna un objeto si y solo si la cadena objetivo coincide en todas su extension con el patrón buscado.

```
>>> #fullmatch()
>>> mensaje = """wgrxm sigyv zmlnn uqgvh ytftd qdyzd ttcgh fgycm ylujp wukxc gdwjc mjkmm tcxxf rvapd deykj ahrjj sqyil t
tzfe gggla khejr tqoqk pmapf pdgoq zcqho qttpx apdro fdoul uwbnd fwuvs gwvdy jloob nqoam wjeba eihlb gteph hgpst qdmat c
cffw yvhvu x"""
>>> print(re.fullmatch(r".{5}",mensaje))
None
>>> print(len(mensaje))
235
>>> print(re.fullmatch(r'.{235}',mensaje))
<re.Match object; span=(0, 235), match='wgrxm sigyv zmlnn uqgvh ytftd qdyzd ttcgh fgycm y>
>>>
```

split()

separa una cadena en cada posición donde encuentra el patron especificado y retorna una lista con los elementos separados.

```
>>> print(mensaje)
wgrxm sigyv zmlnn uqgvh ytftd qdyzd ttcgh fgycm ylujp wukxc gdwjc mjkmm tcxxf rvapd deykj ahrjj sqyil ttzfe gggla khejr
tqoqk pmapf pdgoq zcqho qttpx apdro fdoul uwbnd fwuvs gwvdy jloob nqoam wjeba eihlb gteph hgpst qdmat ccffw yvhvu x
>>> mensaje_separado = re.split(r"\s",mensaje)
>>> print(mensaje_separado)
['wgrxm', 'sigyv', 'zmlnn', 'uqgvh', 'ytftd', 'qdyzd', 'ttcgh', 'fgycm', 'ylujp', 'wukxc', 'gdwjc', 'mjkmm', 'tcxxf', 'r
vapd', 'deykj', 'ahrjj', 'sqyil', 'ttzfe', 'gggla', 'khejr', 'tqoqk', 'pmapf', 'pdgoq', 'zcqho', 'qttpx', 'apdro', 'fdou
l', 'uwbnd', 'fwuvs', 'gwvdy', 'jloob', 'nqoam', 'wjeba', 'eihlb', 'gteph', 'hgpst', 'qdmat', 'ccffw', 'yvhvu', 'x']
>>>
```

En este ejemplo separamos la cadena “mensaje” en toda ubicación donde encuentre espacios.

Modificadores (Flags o Banderas)

Los modificadores permiten modificar aspectos en el comportamiento de las expresiones regulares(regex), estos pueden ser mencionados con su nombre completo o con la abreviación de su primera letra en mayúscula, son útiles cuando deseamos ignorar mayúsculas y minúsculas por mencionar un ejemplo.

Si deseamos utilizar mas de un modificador podemos usar el operador “|” OR.

Ejemplo: re.I | re.M o re.IGNORECASE | re.ASCII

IGNORECASE,I

No tiene en cuenta mayúsculas ni minúsculas

```
>>> import re
>>> cadena = "Jesus HERNANDEZ"
>>> print(re.search(r"hernandez",cadena))
None
>>> print(re.search(r"hernandez",cadenas,re.IGNORECASE))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'cadenas' is not defined. Did you mean: 'cadena'?
>>> print(re.search(r"hernandez",cadena,re.IGNORECASE))
<re.Match object; span=(6, 15), match='HERNANDEZ'>
>>>
```

Se observa como en el primer intento de búsqueda retorna None debido a que busca hernandez en minúsculas, en el segundo intento agregamos la bandera IGNORECASE y entonces realiza la búsqueda del patron sea mayúscula o minúscula.

DOTALL,S

Permite que el metacaracter punto(.), tenga en cuenta las líneas en blanco.

```
>>> cadena = "ML\nand AI"
>>> print(re.search(r".+", cadena))
<re.Match object; span=(0, 2), match='ML'>
>>> print(re.search(r".+",cadena, DOTALL))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'DOTALL' is not defined
>>> print(re.search(r".+",cadena, re.DOTALL))
<re.Match object; span=(0, 9), match='ML\nand AI'>
>>>
```

En este ejemplo estamos buscando el patron ".+" que significa cualquier carácter excepto el que represente una nueva línea, sin embargo, la barra reversa que encontramos en la cadena "ML\nand AI" no representa una nueva línea, por lo tanto, si no especificamos DOTALL solo extrae los caracteres "ML", como se puede ver en el primer intento.

MULTILINE,M

Sirve para utilizar los caracteres de inicio (^) y fin ($) de línea en una cadena con más de una línea.

Esta bandera se utiliza con los metacaracteres **^** cuando se quiere encontrar un patron al inicio de la cadena y al inicio de una nueva línea, y **$** cuando se desea encontrar el patron al final de la cadena y al final de cada nueva línea.

```
>>> import re
>>> cadena =  "Joy lucky number is 75\nTom lucky number is 25"
>>> # encontraremos 3 letras al inicio de cada linea sin usar la bandera
>>> print(re.findall(r"^\w{3}",cadena))
['Joy']
>>> #encontraremos 2 digitos al final de cada linea sin usar la bandera
>>> print(re.findall(r"\d{2}$",cadena))
['25']
>>> #usaremos la bandera para encontrar las tres primeras letras al inicio de cada linea
>>> print(re.findall(r"^\w{3}", cadena, re.MULTILINE))
['Joy', 'Tom']
>>> #usaremos la bandera para encontrar 2 digitos al final de cada linea
>>> print(re.findall(re"\d{2}$",cadena,re.MULTILINE))
  File "<stdin>", line 1
    print(re.findall(re"\d{2}$",cadena,re.MULTILINE))
                        ^^^^^^^^
SyntaxError: invalid syntax
>>> print(re.findall(r"\d{2}$",cadena,re.MULTILINE))
['75', '25']
>>>
```

VERBOSE,X

Hace el patrón de búsqueda de una forma que sea más sencilla de entender y leer permitiendo comentarios.

```
>>> cadena =  "Jesus Hernandez loves to write code in Python, C,  and JavaScript"
>>> print(re.search(r"(^\w{2,}) # encuentra al menos 2 letras al inicio de la cadena",cadena,re.VERBOSE))
<re.Match object; span=(0, 5), match='Jesus'>
>>>
```

En el ejemplo podemos ver comentarios(#) dentro de la función, sin embargo, son ignorados al incluir la bandera VERBOSE.

ASCII,A

Hace que las secuencias de escape \w, \W, \b, \B, \s, \S, \d, \D funcionen para coincidencias con los caracteres ASCII.

```
>>> cadena = "虎太郎 and Jessa are friends"
>>> #Resultado sin usar la bandera
>>> print(re.findall(r"\b\w{3}\b", cadena))
['虎太郎', 'and', 'are']
>>> #Resultado usando la bandera
>>> print(re.findall(r"\b\w{3}\b", cadena, re.ASCII))
['and', 'are']
>>>
```

Métodos de las funciones

Cada función que utilizamos para buscar coincidencias retorna un objeto con los resultados, este objeto a su vez contiene métodos que podemos utilizar para dar forma a estos resultados.

group(): Devuelve la subcadena que coincide con el patron de búsqueda.

start(): Devuelve la posición inicial de la coincidencia.

end(): Devuelve la posición final de la coincidencia.

span(): Devuelve una tupla con la posición inicial y final de la coincidencia.

Ejemplo

```
>>> saludo = re.search(r"o","hola")
>>> saludo.group()
'o'
>>> saludo.start()
1
>>> saludo.end()
2
>>> saludo.span()
(1, 2)
>>>
```

Ejemplo de interacción con usuario

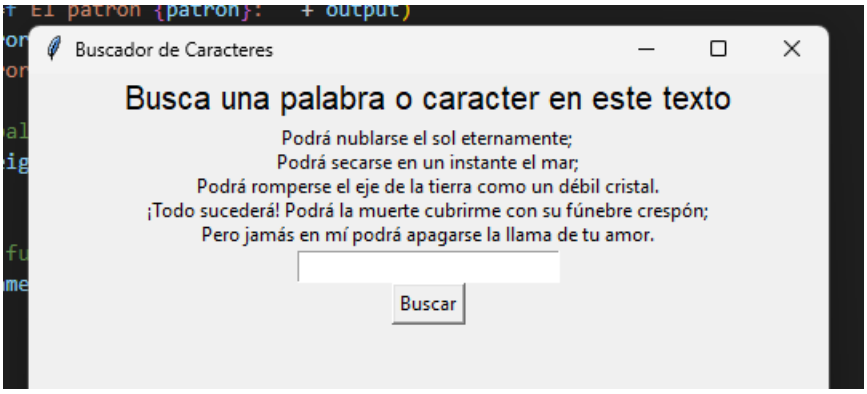
Interfaz del proyecto realizada con la librería **tkinter** de Python, el objetivo es buscar una palabra o carácter dentro de una cadena de texto proporcionada, para realizar la búsqueda se aplica lo aprendido con el **módulo re** de Python y las **regex.**

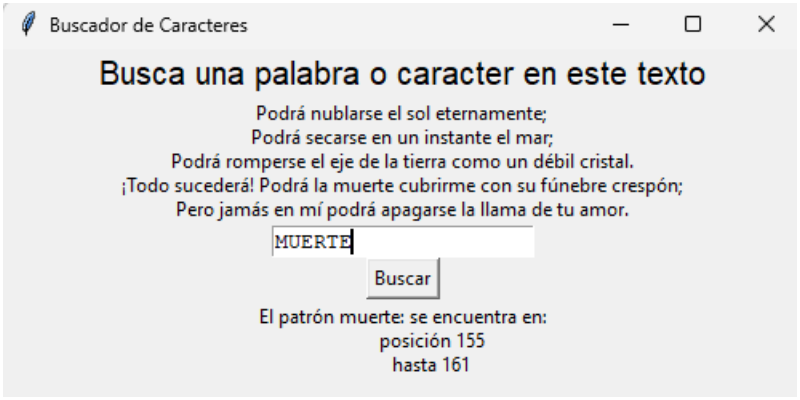
Código

```
import tkinter as tk
from tkinter.messagebox import showerror
import re

# Configuración de la pantalla interfaz
frame = tk.Tk()
frame.title("Buscador de Caracteres")
frame.geometry('500x500')

subtitulo = tk.Label(frame, text="Busca una palabra o caracter en este texto",font=("Helvetica",15))
subtitulo.pack()
# Cadena de texto donde se desea buscar el caracter, palabra o subcadena
cadena = tk.Label(frame, text="""Podrá nublarse el sol eternamente;
Podrá secarse en un instante el mar;
```

```
Podrá romperse el eje de la tierra como un débil cristal.
¡Todo sucederá! Podrá la muerte cubrirme con su fúnebre crespón;
Pero jamás en mí podrá apagarse la llama de tu amor.""" )
cadena.pack()

# Función que buscará la palabra usando la librería re
def buscar():
  inp = inputtxt.get(1.0, "end-1c")
  try:
    match = re.search(f"{inp}", cadena.cget("text"), re.IGNORECASE)
    print(match)
    if(match == None):
      lbl.config(text=f"El patrón {inp} no fue localizado")
    else:
      patron = match.group()
      print(patron)
      posicion = match.span()
      output = f"""se encuentra en:
      posición {posicion[0]}
      hasta {posicion[1]}"""
      lbl.config(text=f"El patrón {patrón}: " + output)
  except ValueError as error:
    showerror(title='Error', message=error)

# TextBox para capturar la palabra o caracter que se desea buscar
inputtxt = tk.Text(frame, height=1, width=20)
inputtxt.pack()

# Botón buscar ejecutará la función buscar() al hacer clic
boton_buscar = tk.Button(frame, text="Buscar", command=buscar)
boton_buscar.pack()

# Salida resultado
lbl = tk.Label(frame, text="")
lbl.pack()

frame.mainloop()
```

Interfaz





Bibliografia

Fernandez Montoro, A. (2013). *Python 3 al descubierto.* Mexico D.F: Alfaomega Grupo Editor S.A de C.V .

Hinojosa Gutierrez, A. (2015). *Python paso a paso.* Madrid: RA-MA editorial.

How to Get the Input From Tkinter Text Box? (11 de 12 de 2020). Obtenido de geeksforgeeks.org: https://www.geeksforgeeks.org/how-to-get-the-input-from-tkinter-text-box/#

Lopez Briega, R. E. (19 de 07 de 2015). *Matemáticas, análisis de datos y python*. Obtenido de relopezbriega.github.io: https://relopezbriega.github.io/blog/2015/07/19/expresiones-regulares-con-python/

Lujan Castillo, J. D. (2020). *Aprende a programar con Python.* Ciudad de Mexico: Alfaomega Grupo Editor S.A de C.V.

Tkinter Example. (s.f.). Obtenido de Pyhton Tutorial: https://www.pythontutorial.net/tkinter/tkinter-example/

Vishal, H. (2 de 4 de 2021). *Python Regex Match: A Comprehensive guide for pattern matching*. Obtenido de PYnative Python Programming: https://pynative.com/python-regex-pattern-matching/