Instituto Politécnico Nacional

Escuela Superior de Cómputo

Teoría Computacional

Práctica 3

Alumno: Meléndez Padilla Mauricio

Profesor: Rosas Trigueros Jorge Luis

2CV4

Fecha de realización de la práctica

27/SEP/18

Fecha de entrega del reporte

6/OCT/18

**Marco Teórico**

**Expresiones regulares en Linux**

Una expresión regular es un patrón que nos permite buscar un texto formado por metacaracteres y caracteres ordinarios.  
  
Los metacaracteres son ciertos caracteres con un significado específico dentro de una expresión regular. Estos caracteres tienen un significado que va más allá del símbolo que representan y tienen un comportamiento especial en una expresión regular.  
  
Aquí tenéis una lista de metacaracteres que usamos en expresiones regulares:

* . Significa cualquier caracter.
* ^Indica el principio de una línea.
* $ Indica el final de una línea.
* \* Indica cero o más repeticiones del caracter anterior.
* + Indica una o más repeticiones del caracter anterior.
* \< Indica el comienzo de una palabra.
* \> Indica el final de una palabra.
* \ Caracter de escape. Da significado literal a un metacaracter.
* [ ] Uno cualquiera de los caracteres entre los corchetes. Ej: [A-Z] (desde A hasta Z).
* [^ ] Cualquier caracter distinto de los que figuran entre corchetes: Ej: [^A-Z].
* { } Nos permiten indicar el número de repeticiones del patrón anterior que deben darse.
* | Nos permite indicar caracteres alternativos: Ej: (^|[?&])
* ( ) Nos permiten agrupar patrones. Ej: ([0-9A-F]+:)+

En las expresiones regulares se distingue entre mayúsculas y minúsculas.  
Si queremos representar un caracter entre la a y la z, lo haremos de la siguiente manera: [a-z]  
Dentro del conjunto, podemos especificar todos los caracteres que queramos. Por ejemplo: [a-zA-Z] representaría los caracteres alfabéticos en minúsculas y mayúsculas. Eso sí. El conjunto representa a un sólo caracter.  
Si lo que queremos es representar identificar un número o una letra, podríamos hacerlo así:  
[a-zA-Z0-9]  
  
Los conjuntos pueden representarse, nombrando todos y cada uno de los elementos, o el intervalo. Ej: [0-9] representa lo mismo que [0123456789].

ejemplos de expresiones regulares:  
  
**# grep '^La' fichero**  
El comando anterior nos devuelve todas las líneas del fichero que comienzan por **La**.  
  
**# grep '^ \*La' fichero**  
El comando anterior nos devuelve todas las líneas del fichero que comienzan por cualquier número de espacios seguido de **La**.  
  
**# grep '^\..\*' fichero**  
El comando anterior nos devuelve todas las líneas del fichero que comienzan por punto y tienen cualquier número de caracteres.  
  
**# ls -la | grep ' \..\*'**  
El comando anterior nos devuelve la lista de ficheros que comienzan por un espacio seguido de un punto y cualquier número de caracteres, es decir, la lista de ficheros ocultos.  
  
**# ls -l | grep '^d'**  
El comando anterior nos devuelve la lista de ficheros que comienzan por **d**, es decir, la lista de directorios.

**Material y equipo.**

El material utilizado en la práctica es el siguiente:

Herramientas de software:

• Mac OS X 10.13.6

• Python 2.7.15

• VIM - Vi IMproved 8.1

• Terminal

Herramientas de hardware:

• Computadora personal.

**Desarrollo de la práctica.**

1. Resuelva los ejercicios de expresiones regulares de la página <https://regex.sketchengine.co.uk/>
2. Resolver la guía para practicar con expresiones regulares:

* Fecha
* E-mail
* CURP
* RFC
* Dirección IP
* URL

Código:

*#!/usr/bin/env python*

*# -\*- coding: utf-8 -\*-*

import re

patron = re.compile(r'\bfoo\b') *# busca la palabra foo*

texto = """ bar foo bar

foo barbarfoo

foofoo foo bar

"""

print(patron.match(texto))

m = patron.match('foo bar')

m

s = patron.search(texto)

s

fa = patron.findall(texto)

fa

fi = patron.finditer(texto)

fi

next(fi)

next(fi)

m.group(), m.start(), m.end(), m.span()

s.group(), s.start(), s.end(), s.span()

mail = re.compile(r"""

\b *# comienzo de delimitador de palabra*

[\w.%+-] *# usuario: Cualquier caracter alfanumerico mas los signos (.%+-)*

+@ *# seguido de @*

[\w.-] *# dominio: Cualquier caracter alfanumerico mas los signos (.-)*

+\. *# seguido de .*

[a-zA-Z]{2,6} *# dominio de alto nivel: 2 a 6 letras en minúsculas o mayúsculas.*

\b *# fin de delimitador de palabra*

""", re.X)

mails = """raul.lopez@relopezbriega.com, Raul Lopez Briega,

foo bar, relopezbriega@relopezbriega.com.ar, raul@github.io,

https://relopezbriega.com.ar, https://relopezbriega.github.io,

python@python, river@riverplate.com.ar, pythonAR@python.pythonAR

"""

mail.findall(mails)

url = re.compile(r"^(https?:\/\/)?([\da-z\.-]+)\.([a-z\.]{2,6})([\/\w \.-]\*)\*\/?$")

*# vemos que https://relopezbriega.com.ar lo acepta como una url válida.*

url.search("https://relopezbriega.com.ar")

print(url.search("https://google.com/un/archivo!.html"))

patron = ('^(?:(?:25[0-5]|2[0-4][0-9]|'

'[01]?[0-9][0-9]?)\.){3}'

'(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$')

ip = re.compile(patron)

*# la ip 73.60.124.136 es valida*

ip.search("73.60.124.136")

print(ip.search("256.60.124.136"))

fecha = re.compile(r'^(0?[1-9]|[12][0-9]|3[01])/(0?[1-9]|1[012])/((19|20)\d\d)$')

*# validando 13/02/1982*

fecha.search("13/02/1982")

print(fecha.search("13-02-1982"))

print(fecha.search("32/12/2015"))

print(fecha.search("30/14/2015"))

*#RFC*

RFC=re.compile(r'/^([A-ZÑ&]{3,4}) ?(?:- ?)?(\d{2}(?:0[1-9]|1[0-2])(?:0[1-9]|[12]\d|3[01])) ?(?:- ?)?([A-Z\d]{2})([A\d])$/')

print(RFC.search('MEPM9404182C2'))

*#CURP*

curp=re.compile(r'/^([A-ZÑ&]{3,4}) ?(?:- ?)?(\d{2}(?:0[1-9]|1[0-2])(?:0[1-9]|[12]\d|3[01])) ?(?:- ?)?([A-Z\d]{2})([A\d])$/')

print(curp.search('MEPM940418HDFLDR08'))

**Diagramas, gráficas y pantallas**

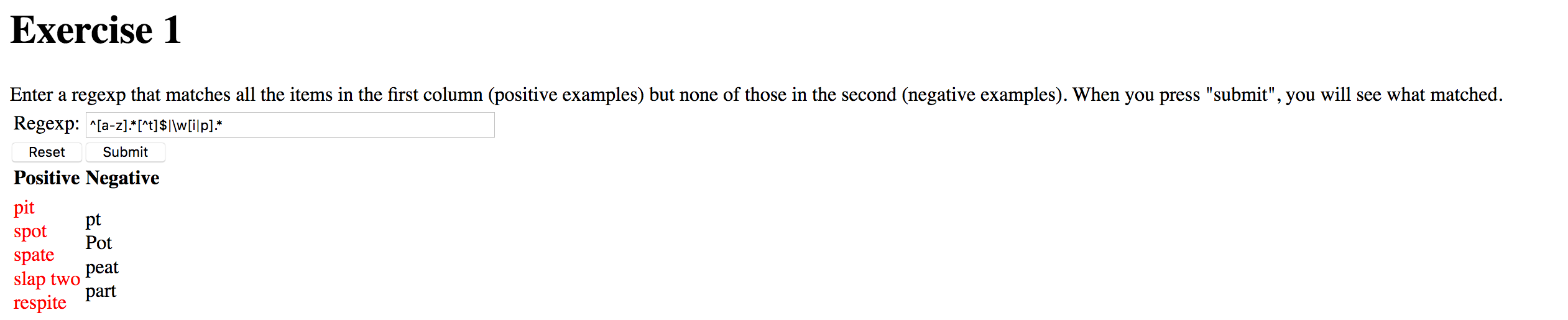


Imagen 6.1 Ejercicio 1

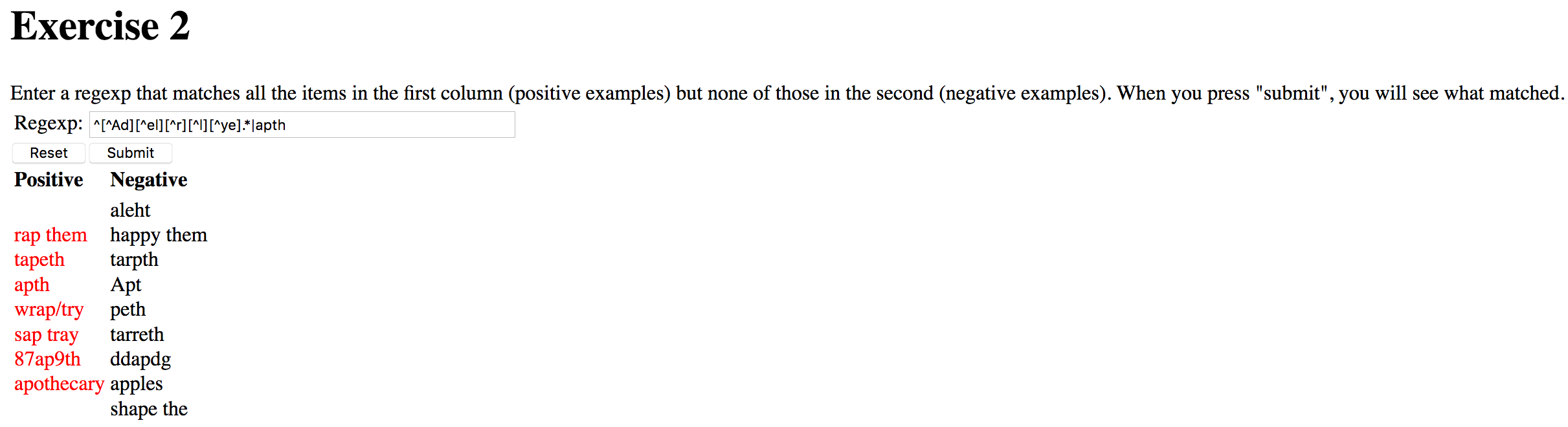
****

Imagen 3.2 Ejercicio 2

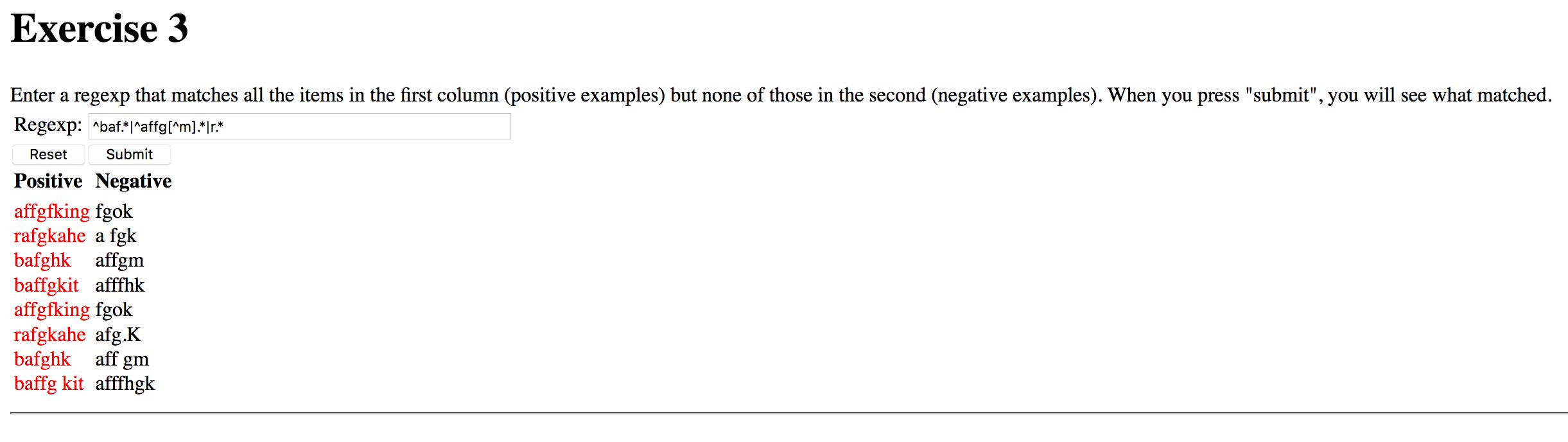
****

Imagen 3.3 Ejercicio 3

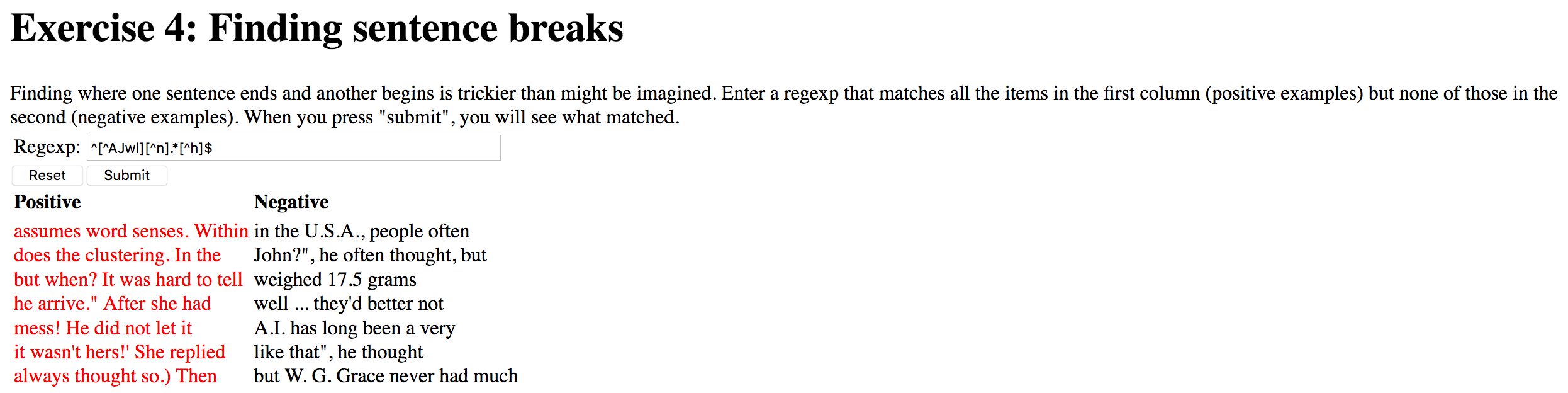
****

Imagen 3.4 Ejercicio 4

**Conclusiones y recomendaciones**

Las expresiones regulares en Linux y Python, así como en los diversos lenguajes de programación son muy útiles a la hora de encontrar ocurrencias con una dicha búsqueda, no siempre se puede buscar algo en concreto, sin embargo las expresiones regulares nos permiten todo tipo de cosas y posibilidades a la hora de las búsquedas.

**Bibliografía**

* [1]"Expresiones Regulares En.wikipedia.org, 2018. [Online]. Available: https://es.wikipedia.org/wiki/Expresi%C3%B3n\_regular. [Accessed: 4-OCT-2018].
* [2]"Expresiones Regulares con Python, 2018. [Online]. Available: https://relopezbriega.github.io/blog/2015/07/19/expresiones-regulares-con-python/ . [Accessed: 4-OCT-2018]