UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Centro de e-Learning

UNIDAD DIDÁCTICA VII

DIPLOMATURA EN PYTHON

Centro de e-Learning SCEU UTN - BA.
Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Módulo II – Nivel Inicial II

Unidad VII – Expresiones regulares II.

Bloques temáticos:

1.- Beautifulsoup y regex.

2.- Otros lenguajes y regex.

3.- Lanzar sitio web desde python.

4.- Ejemplos con expresiones regulares.

1.- Beautifulsoup y regex.

Beautiful Soup es una biblioteca de Python para extraer datos de archivos HTML y XML. Funciona con su analizador favorito para proporcionar formas idiomáticas de navegar, buscar y modificar el árbol de análisis. Generalmente ahorra a los programadores horas o días de trabajo.

Instalación

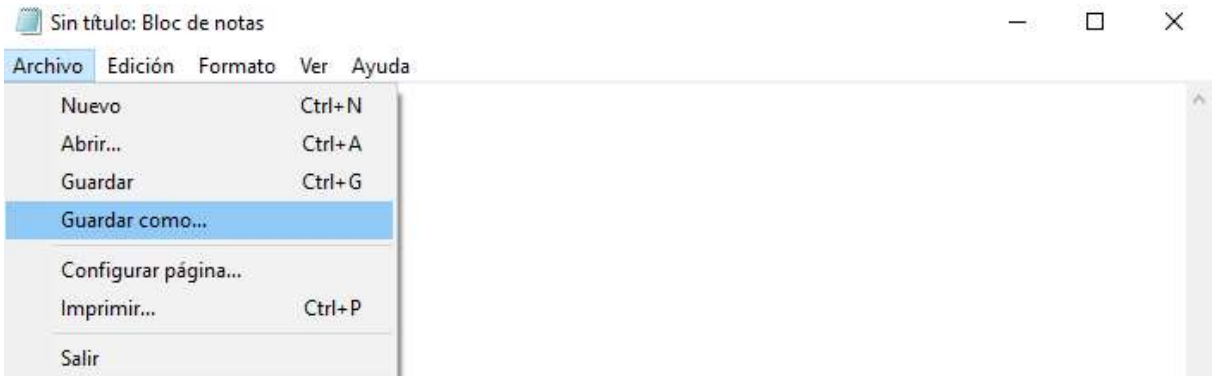
Abrimos el cmd y ejecutamos:

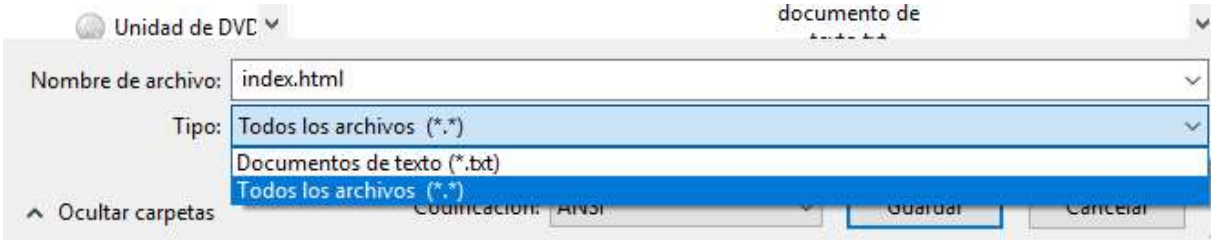
```
pip install beautifulsoup4
pip install requests
```

Creamos nuestra primera página web.

Para crear nuestra primer página web solo necesitamos un editor de texto y para mostrar que en realidad lo podemos hacer con cualquiera, vamos a utilizar el más simple de todos en Windows que es el Bloc de notas:

Por lo que simplemente lo abro y guardo el archivo que se abre como index.html.





Ojo en este editor para guardar un formato diferente a txt debemos indicar "Todos los archivos (*.*)"

Si ves el icono del archivo que se ha creado, notarás que el sistema operativo reconoce que se trata de una página web ya que lo abre con el explorador configurado por defecto en el sistema:

Ahora agregamos el siguiente contenido en la misma y guardamos los cambios:

index.html

```
<!Doctype html>
<html lang="es">
        <head>
           <meta charset="utf-8">
           <title>Mi Primera página Web</title>
        </head>
        <body>
                <h1>Hola!</h1>
        </body>
<html>
```

Lo que acabamos de escribir es código html, en particular html5, **¿Pero qué significa?**

Html es un lenguaje de etiquetas, las cuales son nombres predefinidos que se escriben entre el signo de menor y mayor en esta forma:

```
<!Doctype html>
```

Esta es la primera etiqueta que indica que lo que viene a continuación es código html5, luego tenemos la etiqueta html que es la que engloba todo el código que escribamos de aquí en adelante, de hecho como html es un caso particular del lenguaje XML sigue las mismas reglas que este y una de las primeras reglas es que debe existir una etiqueta que englobe todo el contenido, la cual se llama etiqueta raíz:

index.html

```
<!Doctype html>
<html lang="es">
<html>
```

Se llama etiqueta raíz ya que el documento html es entendido por cada explorador como las raíces de un árbol que se va extendiendo desde un tronco principal o raíz principal. Esta forma de comprender el documento web se llama DOM (Document Object Model) o en español 'Modelo en Objetos para la Representación de Documentos'. Para el explorador cada etiqueta es comprendida como un nodo del cual se desprenden raíces secundarias conteniendo cada una otros nodos. Sigamos avanzando y veamos cómo es este tipo de representación. Las siguientes etiquetas agregadas son <head> y <body>, ambas poseen una etiqueta de apertura y de cierre.

index.html

```
<!Doctype html>
<html lang="es">
        <head>

        </head>
        <body>

        </body>
<html>
```

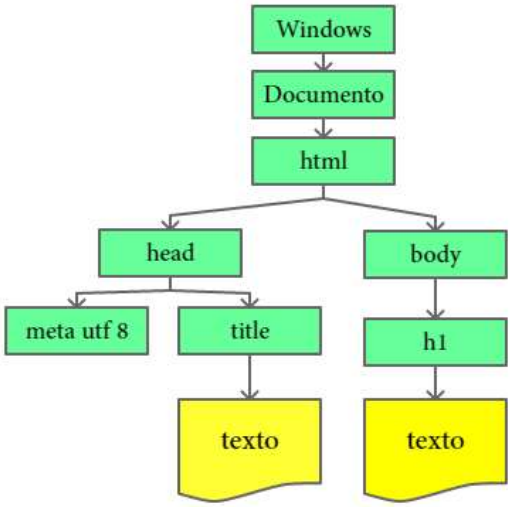
La etiqueta <head> es utilizada para agregar todas las cabeceras, como rutas a otros archivos utilizados dentro de este archivo, título de la página, estilos, javascript, etc, mientras que <body> se utiliza como contenedor de todo lo que aparece en nuestra página web, por lo que si por ejemplo queremos agregarle a la página una imagen, un título, un artículo o cualquier otro tipo de elemento, éste debe ir dentro de la etiqueta <body>. Finalmente agregamos una etiqueta para el título y otra para poder utilizar texto

en español dentro de la etiqueta <head>, así como una etiqueta de título dentro del <body> el cual visualizamos en pantalla.

index.html

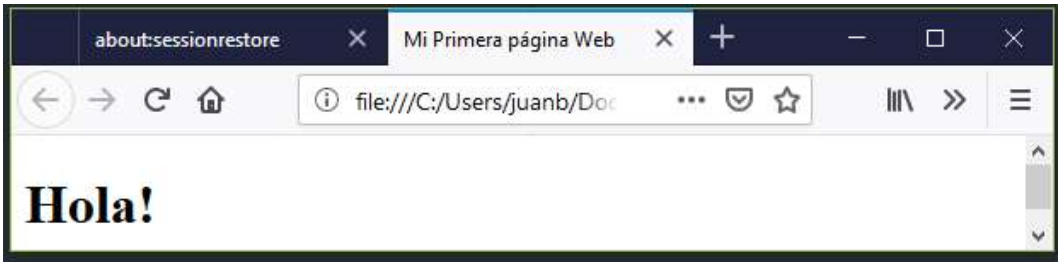
```
<!Doctype html>
<html lang="es">
        <head>
           <meta charset="utf-8">
           <title>Mi Primera página Web</title>
        </head>
        <body>
                <h1>Hola!</h1>
        </body>
<html>
```

La representación de nuestra primera página web desde el punto de vista del DOM sería como sigue:



Cada uno de los elementos representados es lo que llamamos un nodo, notar que incluso el texto es comprendido como un nodo. Para realizar el esquema anterior he utilizado la herramienta de prototipos de uso libre “Pencil” la cual se puede descargar y utilizar de forma muy amigable.

Si ahora con los cambios guardados en nuestro archivo nos paramos sobre éste y hacemos doble click con el botón izquierdo del mouse, podemos ver que se abre un explorador mostrándonos la página web creada:



Utilizamos beautifulsoup con texto html

Para comenzar a utilizar beautifulsoup, importamos la librería y tomaremos el código de la página web que acabamos de crear y lo guardaremos dentro de un objeto "soup" utilizando el método "BeautifulSoup()", luego el método prettify () convertirá el árbol de análisis de Beautiful Soup en una cadena Unicode bien formateada, con cada etiqueta HTML / XML en su propia línea:

beautifulsoup1.py

```
from bs4 import BeautifulSoup
soup = BeautifulSoup("<!Doctype html><html><head></head><body>
<h1>Hola!</h1></body></html>", "html.parser")
soup.prettify()
print(soup.prettify())
```

Al ejecutar el código nos retorna:

```
<!Doctype html>
<html lang="es">
  <head>
    <meta charset="utf-8">
    <title>Mi Primera página Web</title>
  </head>
  <body>
    <h1>Hola!</h1>
```

```
  </body>
<html>
```

Navegar html

Veamos ahora cómo recorrer un documento html, partamos de guardar dentro de una variable una página web.

beautifulsoup2.py

```
documento_html="""
<!Doctype html><html><head><title>Recorrer Documento html</title></head>
<body>
<p class ="titulo"><b>Título de la historia</b></p>
<p class = “contenido”>Contenido de la historia
<a href="ruta 1" class = "estiloAncla" id="link1"> historia 1</a>
<a href="ruta 1" class = "estiloAncla" id="link2"> historia 2</a>
<a href="ruta 1" class = "estiloAncla" id="link3"> historia 3</a>
<a href="ruta 1" class = "estiloAncla" id="link4"> historia 4</a>
</p>
<p class ="historia">……..</p>
"""
```

Cómo vimos antes, el contenido lo podemos recuperar así:

```
soup = BeautifulSoup(documento_html, "html.parser")
print(soup.prettify())
```

Pero podríamos acceder al DOM y recuperar una etiqueta en particular con todo su contenido de la siguiente forma:

```
print(soup.title, end="\n--------1---------\n")
print(soup.head, end="\n---------2--------\n")
print(soup.body, end="\n---------3--------\n")
print(soup.body.b, end="\n-------4----------\n")
print(soup.a, end="\n---------5--------\n")
```

Con estas simples líneas de código hemos recuperado las etiquetas <title>, <head>, <body>, <b> y <a>.

Nota: En el caso de la etiqueta <b> estamos siendo específicos y con notación de punto estamos indicando que se trata de la etiqueta <b> que se encuentra dentro de la etiqueta <body>.

Nota: En todos los casos recupera la primera coincidencia. Esto lo podemos ver evidenciado cuando recuperamos la etiqueta <a> ya que solo nos trae la primera. Si queremos recuperar todas las etiquetas <a> podemos utilizar el método "find_all()" el cual guardará cada etiqueta <a> recuperada junto con su contenido dentro de un elemento de lista, por lo que sí quiero recuperar todas las etiquetas, pero mostrar solo la primera lo puedo hacer de la siguiente forma.

```
soup.find_all('a')
array = soup.find_all('a')
print(array[0], end="\n--------6---------\n")
```

O también podría utilizar un bucle para recorrer y recuperar cada elemento.

```
body_tag = soup.body
for i in body_tag.children: print(i)
```

Aquí al seguir con notación de punto a la etiqueta con la palabra "children", estamos haciendo referencia a las etiquetas hijas.

Uso de expresiones regulares.

Ahora que tenemos una idea de que es un documento html y de cómo utilizar beautifulsoup, podemos adicionar nuestro conocimiento de expresiones regulares para generar consultas de análisis más complejas y útiles.

```
import re
for tag in soup.find_all(re.compile("^b")): print(tag.name)
for tag in soup.find_all(re.compile("t")): print(tag.name)
```

Como podemos imaginar con el código anterior estamos recuperando todas las etiquetas que inician con "b" o que contienen la letra "t".

Accediendo a html externo.

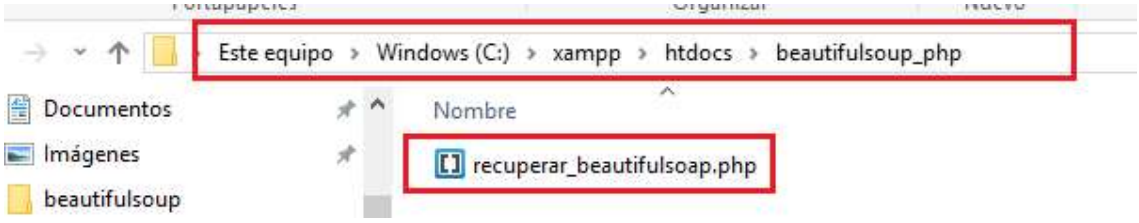
Estamos en condiciones ahora de leer un documento html externo, hay que recalcar que el documento web está realizado en html, este lenguaje es el que permite visualizar texto en un explorador web, sin embargo solo permite agregar información estática, es decir que con html no podemos por ejemplo conectarnos con una base de datos o crear un carrouselle, para ello debemos utilizar otro lenguaje de programación como python, php, javascript, java, ruby, c#, etc. Es muy interesante notar que muchas veces tenemos que conectarnos con un servidor que se encuentra programado en un lenguaje de programación que puede no ser python, por ejemplo en el mundo web la mayoría trabaja con php ya que es un lenguaje con un costo de aprendizaje mucho menor que python. Para mostrar que la comunicación la podemos realizar entre diferentes lenguajes, vamos a crear dos páginas web en php y nos comunicaremos con ellas, para extraerles información.

Caso 1: Leyendo archivo externo

Prendamos el servidor web que descargamos junto con la base de datos de MySQL, el mismo lo podemos encender desde la ventana emergente que utilizamos para acceder a phpmyadmin.

Nota: Podemos ver que el servidor web local se inicia en el puerto 80, el cual es utilizado por las páginas web.

Para mostrar una página web mediante un servidor, debemos agregar la página a mostrar dentro de la carpeta pública htdocs que se encuentra dentro de xampp, en esta carpeta crearemos un directorio al cual llamaremos “beautifulsoap_php” y dentro crearemos un archivo del tipo php llamado “recuperar_beautifulsoap.php”

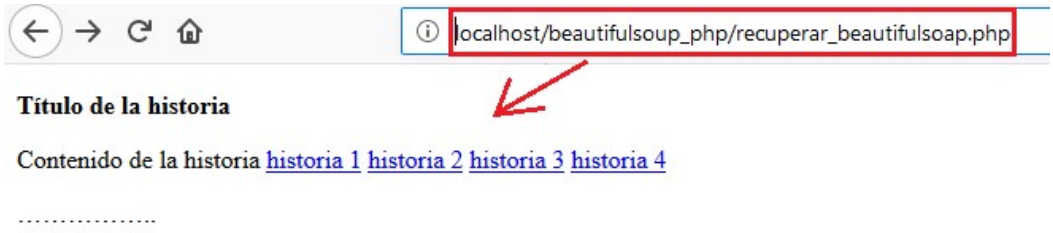


recuperar_beautifulsoap.py

```
<!Doctype html>
<html>
        <head>
                <title>Beautiful soup recorrer html</title>
        </head>
        <body>
                <p class ="titulo"><b>Título de la historia</b></p>
                <p class = "contenido">Contenido de la historia
                <a href="ruta 1" class = "estiloAncla" id="link1"> historia 1</a>
                <a href="ruta 1" class = "estiloAncla" id="link2"> historia 2</a>
                <a href="ruta 1" class = "estiloAncla" id="link3"> historia 3</a>
                <a href="ruta 1" class = "estiloAncla" id="link4"> historia 4</a>
                </p>
                <p class ="historia">……..</p>
        </body>
</html>
```

Como podemos ver en el código anterior, la extensión del archivo es ".php" pero el contenido no difiere del que hemos agregado al ver un archivo del tipo "html". Esto se debe a que trabajar con un archivo del tipo ".php" es idéntico a trabajar con un archivo del tipo ".html" salvo que en el de tipo ".php" podemos programar y en el ".html" no.

Si ahora accedemos a un explorador y escribimos la ruta desde el servidor (el cual por defecto se llama "localhost") podemos ver el contenido de la página en el explorador.



Este contenido lo podemos extraer desde python, utilizando el siguiente código:

beautifulsoap3.py
```
import requests
from bs4 import BeautifulSoup

data = requests.get("http://localhost/beautifulsoup_php/recuperar_beautifulsoap.php")
print(data)
soup = BeautifulSoup(data.text, "html.parser")
print(soup.find('a',{'class':'estiloAncla'}))
```

El cual nos retorna el contenido recuperado y mediante un print nos permite ver la primer etiqueta del tipo <a> con su contenido.

```
<Response [200]>
<a class="estiloAncla" href="ruta 1" id="link1"> historia 1</a>
```

Caso 2: Pasando información a archivo externo y recuperando contenido.

Algo que puede ser más interesante aún, es pasarle contenido a un archivo externo y que nos retorne un resultado, de esta forma podemos trabajar sobre un programa que en este caso va a ser realizado en php y recuperar el valor desde python. Para ver cómo funciona agregaremos un archivo php más a nuestro servidor local y nos comunicaremos desde un programa en python.

recuperar_beautifulsoap2.php
```
<?php
$variable1 = $_GET['variable1'];
$variable2 = $_GET['variable2'];
echo "El valor de la variable 1 es: " . $variable1 . " y el de la variable 2 es: " . $variable2;
```

En este caso el contenido de la página no posee nada de código html, pues estamos utilizando el archivo para retornar información y no es necesario imprimir nada en la pantalla del explorador, solo estamos utilizando el archivo para comunicarnos entre php y python.

El código php debe agregarse entre etiquetas de inicio “<?php” y de fin “?>”, sin embargo por recomendación del sitio oficial de php cuando el código es puramente php la etiqueta de cierre debe ser omitida.

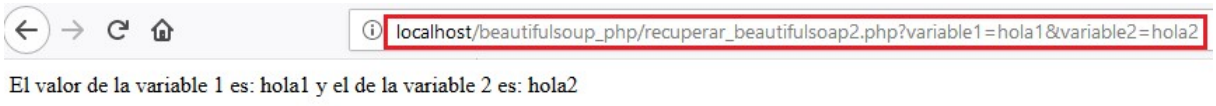
El código es bastante simple se declaran dos variables que mediante “$_GET[‘ ‘]” recuperan los valores que se están enviando a través de la url y se imprime el resultado en pantalla utilizando un “echo” que como valor tiene texto plano que al igual que en python se pone entre comillas y los valores de las variables concatenadas con notación de punto.

Nota: Cada línea de código en php debe llevar al final un punto y coma “;”.

Si en el explorador escribimos:

http://localhost/beautifulsoup_php/recuperar_beautifulsoap2.php?variable1=hola1&variable2=hola2

Podemos ver la información que nos retorna la página.



En la url hemos agregado la primer variable concatenándola con un signo de “?” y de la siguiente en adelante se concatenan con “&”.

Este código lo podemos recuperar desde python mediante el siguiente código.

beautifulsoap4.py

```
import requests
from bs4 import BeautifulSoup
data2 =
requests.get("http://localhost/beautifulsoup_php/recuperar_beautifulsoap2.php?variable1=
hola1&variable2=hola2")
soup2 = BeautifulSoup(data2.text, "html.parser")
print(soup2)
```

2. Otros lenguajes y regex.

Las expresiones regulares las podemos utilizar con todos los lenguajes modernos, un caso muy interesante siguiendo con lo que venimos viendo es trabajar sobre las url de un sitio web y transformarlas en direcciones amigables que sean más legibles por el explorador y no revelen la arquitectura de nuestro sitio. Veamos un poco cómo trabajar con las expresiones regulares sobre el servidor apache el cual cuenta con el módulo de escritura mod_rewrite, este utiliza un motor de reescritura, basado en un analizador de expresiones regulares PCRE, para reescribir las URL solicitadas sobre la marcha.

De forma predeterminada, mod_rewrite asigna una dirección URL a una ruta del sistema de archivos. Sin embargo, también se puede utilizar para redirigir una URL a otra URL, o para invocar un proxy interno de obtención de información.

Un **proxy**, en una **red informática**, es un programa o dispositivo que realiza un en representación de otro, esto es, si una hipotética máquina **A** solicita un recurso a una **C**, lo hará mediante una petición a **B; C** entonces no sabrá que la petición procedió originalmente de **A**. Esta situación estratégica de punto intermedio suele ser aprovechada para soportar una serie de funcionalidades: proporcionar caché, control de acceso, registro del tráfico, prohibir cierto tipo de tráfico, etc.

Su finalidad más habitual es la de **servidor proxy**, que consiste en interceptar las conexiones de red que un cliente hace a un servidor de destino, por varios motivos posibles como seguridad, rendimiento, anonimato, etc. Esta función de **servidor proxy** puede ser realizada por un programa o dispositivo.

mod_rewrite proporciona una manera flexible y potente para manipular las URL con un número ilimitado de reglas.

mod_rewrite funciona sobre la ruta URL completa.

Paso 1. Habilitar mode_rewrite

Vamos a: **apache > conf > httpd.conf**

Buscamos la línea:

LoadModule rewrite_module modules/mod_rewrite.so

En el caso de que esté comentada (con el #), la descomentamos eliminando el (#).

Paso 2. Cómo funciona mod_rewrite

Mod_rewrite permite ingresar una url amigable html estática y acceder al archivo php dinámico menos amigable y obtener la respuesta

Es decir, podemos ingresar una url como:

http://www.sitio.com/producto.html

Esta es pasada al servidor y transformada en:

http://www.sitio.com/producto.php

Paso 3. Primer ejemplo

Directorio: SEO/ejercicio1

Para entender cómo funciona el mod_rewrite de apache, podemos comenzar por un ejercicio simple, en el cual tenemos en el directorio de nuestro sitio web un archivo llamado “contenido.php” el cual queremos que sea encontrado como “contenido.html” con lo cual logramos modificar la extensión del archivo.

contacto.php

```
<?php
        echo '<b>Este es el contenido de un determinado artículo</b>';
```

Para realizar esto, necesitamos crear un archivo .htaccess mediante el uso de un editor de texto cualquiera, con el siguiente contenido.

.htaccess

```
RewriteEngine On

# Vamos a sustituir contenido.php por contenido.html
# La L final entre corchetes simplemente indica la finalización.
RewriteRule ^contenido.html$ contenido.php [L]
```

La primer línea “RewriteEngine On” es obligatoria y debe colocarse siempre al inicio del documento.

Las líneas que comienzan con # son de comentarios.

La regla de reescritura “RewriteRule ^contenido.html$ contenido.php [L]” está dividida en tres argumentos separados por espacios, cuya línea comienza con RewriteRule.

- El primero es ^contenido.html$ contiene la línea de URL que se va a escribir a continuación del dominio del sitio para encontrar el archivo “contenido.php”.
- El segundo contenido.php es el nombre del archivo con la extensión a ser modificada en la url.
- El tercero, formado por la letra [L] al final indica la finalización de la línea de sobre-escritura.

Formas de acceder al archivo:

De esta forma ahora contamos con dos formas de acceder a nuestro archivo. Supongamos que los archivos anteriores se encuentran dentro el directorio SEO/ejercicio1/, para acceder a contacto.php ahora podemos realizarlo de dos formas diferentes.

Como: http:/dominio.com/SEO/ejercicio1/contenido.php

O como: http:/dominio.com/SEO/ejercicio1/contenido.html

En el caso de que estemos trabajando de forma local mediante un servidor como wamp, xampp o lamp, accedemos como:

http:/localhost/SEO/ejercicio1/contenido.php

http:/localhost/SEO/ejercicio1/contenido.html

Paso 4. Segundo ejemplo – Sobre escribir URLs dinámicas

Directorio: SEO/ejercicio2

Pasemos a realizar ahora un ejercicio un poco más complejo e interesante. Supongamos que tenemos que pasarle dos parámetros a un archivo determinado, por ejemplo tomemos el archivo “article.php” y pasémosle un identificador del título del artículo y otro del contenido del artículo.

article.php

```
<?php
echo 'Titulo: '.$_GET['idtitulo'];
echo '<br>';
echo 'Contenido: '.$_GET['idcontenido'];
```

.htaccess

```
RewriteEngine On

# Sobre escribir URLs numericas
RewriteRule ^articulo/Titulo([0-9]*)/Contenido([0-9]*)\.html$
article.php?idtitulo=$1&idcontenido=$2 [L]
```

En este caso la regla de reescritura está dividida en tres argumentos separados por espacios, cuya línea comienza con RewriteRule.El primero es RewriteRule, que indica que a continuación va la regla de reescritura.

- El primero ^articulo/Titulo([0-9]*)/Contenido([0-9]*)\.html$ el cual contiene la URL que se va a escribir a continuación del dominio del sitio.

En ese caso la url está formada por un directorio artículo que no existe en nuestro dominio, seguido de lo que sería un directorio Título que puede adoptar un número entre cero y nueve repetido de cero a más veces, seguido del archivo Contenido que tampoco existe que puede adoptar un número entre cero y nueve repetido cero o más veces con extensión .html

- El segundo article.php?idtitulo=$1&idcontenido=$2 es el nombre del archivo con las variables pasadas via GET con la extensión a ser modificada en la url.

- El tercero, formado por la letra [L] al final indica la finalización de la línea de sobre-escritura.

Formas de acceder al archivo:

Nuevamente tenemos dos formas de acceder a nuestro archivo. Supongamos que los archivos anteriores se encuentran dentro del directorio SEO/ejercicio2/, para acceder lo podríamos hacer como:

http:/dominio.com/SEO/ejercicio2/article.php?idtitulo=1&idcontenido=1

O como: http:/dominio.com/SEO/ejercicio2/articulo/Titulo1/Contenido1.html

En el caso de que estemos trabajando de forma local mediante un servidor como wamp, xampp o lamp, accedemos como:

http:/localhost/SEO/ejercicio2/article.php?idtitulo=1&idcontenido=1

http:/localhost/SEO/ejercicio2/articulo/Titulo1/Contenido1.html

Como podemos apreciar, la segunda forma de escribir la URL es mucho más amigable a buscadores. El resultado en el navegador nos muestra:

Paso 5. Regla de reescritura básica

Como hemos vista hasta ahora en los ejercicios anteriores, la regla de reescritura consta de tres argumentos separados por un espacio.

RewriteRule Pattern Substitution [Flags]

- **Pattern**: Es el contenido que se va a escribirse en la URL a continuación del nombre del dominio, se escribe en el archivo .htaccess mediante una expresión regular.
- **Substitution**: Es la url parte de la URL a ser modificada.
- [Flags]: indica una determinada acción que afecta a la regla de reescritura.

Flags

Cada Flags (con pocas excepciones) tiene una forma corta, tal como CO, así como una forma más larga, tal como cookie. Si bien es más común el uso de la forma corta, se recomienda familiarizarse con la forma larga, para poder recordar su significado y que es lo que se supone que debe hacer. Las banderas no son sensibles a mayúsculas.

Los siguientes son algunos de los Flags que podemos encontrar.

Regla de reescritura	Significado	Descripción
B	Escape backreferences	Permite escapar caracteres
R	Redirecciona	Envía una redirección HTTP
F	Prohibido	Prohíbe acceso a una URL
G	Terminada	Marca una URL como terminada
P	Proxy	Pasa la URL a mod_proxy
L	Ultima	Detiene el proceso cuando llega a esta regla
N	Siguiente	Inicia el proceso de nuevo desde el comienzo
C	Encadena	Enlaza la regla actual con la siguiente
T	Tipo	Fuerza el tipo MIME
NS	Nosubreq	La regla aplica solo si no hay una solicitud interna
NC	Nocase	La URL coincide sin tener en cuenta mayúsculas y minúsculas
QSA	Qsappend	Añade una consulta de cadena a parte de una URL y la reemplaza
PT	Passthrough	Pasa una URL sobreescrita a otro modulo Apache para proceso adicional
S	Skipt	Salta a la siguiente regla
E	Env	Crea una variable de entorno

Ver: ***http://httpd.apache.org/docs/current/rewrite/flags.html***

3. Lanzar sitio web desde python.

Ya que estamos trabajando con páginas web, podemos ver cómo crear y lanzar una desde python utilizando el módulo “webbrowser”. Como podemos ver en el siguiente ejemplo podemos crear directamente en el archivo la página html y lanzarla posteriormente ayudándonos del módulo os, el cual ya hemos visto para ubicar el archivo creado en nuestro sistema operativo.

crearhtml.py

```
import os
import webbrowser

f = open('holamundo.html','w')

mensaje = """<!Doctype html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Mi sitio web</title>
        <style>
            p{background-color:aqua;}
        </style>
    </head>
    <body>
        <p>Mi sitio web!</p>
    </body>
</html>
"""

f.write(mensaje)
f.close()
nombreArchivo = os.path.dirname(__file__)+'/'+'holamundo.html'
print(nombreArchivo)
webbrowser.open_new_tab(nombreArchivo)
```

4. Ejemplos con expresiones regulares.

Ejemplo 1

Supongamos que estamos escribiendo un programa de pocker donde la mano de un jugador se representa como una cadena de 5 caracteres con cada personaje que representa una carta, "a" para as, "k" para rey, "q" para reina, "j" para jack, "T" para 10, y "2" a "9" que representan la tarjeta con ese valor.

Creemos una función para encontrar coincidencias:

ejemplo1.py

```
import re

def mostrarCoincidencia(match):
    if match is None:
        return None
    return '<Match: %r, groups=%r>' % (match.group(), match.groups())
```

Para ver si una cadena dada es una mano válida, uno podría hacer lo siguiente:

```
valido = re.compile(r"^[a2-9tjqk]{5}$")
print(mostrarCoincidencia(valido.match("akt5q")))  # Válido.
print(mostrarCoincidencia(valido.match("akt5e")))  # Inválido.
print(mostrarCoincidencia(valido.match("akt")))    # Inválido.
print(mostrarCoincidencia(valido.match("727ak")))  # Válido.
```

Nos retorna:

```
<Match: 'akt5q', groups=()>
None
None
<Match: '727ak', groups=()>
```

Esa última mano, "727ak", contenía un par, o dos cartas iguales. Para hacer coincidir esto con una expresión regular, uno podría usar referencias inversas como:

```
par = re.compile(r".*(.).*\1")
print(mostrarCoincidencia(par.match("717ak"))) # Par de 7s.
print(mostrarCoincidencia(par.match("718ak"))) # No pairs.
print(mostrarCoincidencia(par.match("354aa"))) # Par de ases.
```

Nota: El \1 está haciendo referencia al contenido del grupo (.) con lo que la expresión regular busca cualquier carácter, que puede tener delante cualquier carácter o no y puede estar seguido de cualquier carácter o no y luego \1 hace referencia al mismo carácter de la búsqueda. Con lo que estoy considerando que puede estar repetido.

match.groups (por defecto = Ninguno)

Devuelve una tupla que contiene todos los subgrupos de la coincidencia.

```
<Match: '717', groups=('7',)>
None
<Match: '354aa', groups=('a',)>
```

Ejemplo 2 - Agenda telefónica

Split () divide una cadena en una lista delimitada por el patrón pasado. El método es invaluable para convertir datos textuales en estructuras de datos que Python puede leer y modificar fácilmente, como se muestra en el siguiente ejemplo que crea una guía telefónica. Crearemos primero un texto conteniendo toda la información como si proviniera de un archivo externo y lo guardaremos dentro de la variable “text”, luego extraeremos los datos de cada persona dentro de una tupla.

ejemplo2.py

```
import re

texto = """Juan Perez: 15433476 Rivadavia 3456
Rosita Bonita 11329876 Uruguay 454
Elva Luarte 11098567 Concordia 321
"""
entradas = re.split("\n+", texto)
```

```
print(entradas)
```

Retorna:

```
['Juan Perez: 15433476 Rivadavia 3456', 'Rosita Bonita 11329876 Uruguay 454', 'Elva Luarte 11098567 Concordia 321', '']
```

Finalmente, dividimos cada entrada en una lista con nombre, apellido, número de teléfono, calle y número. En este caso estamos separando por ":" o por espacio en blanco.

```
print([re.split(":? ", entrada) for entrada in entradas])
```

Lo cual nos retorna

```
[['Juan', 'Perez', '15433476', 'Rivadavia', '3456'], ['Rosita', 'Bonita', '11329876', 'Uruguay', '454'], ['Elva', 'Luarte', '11098567', 'Concordia', '321'], ['']]
```

Si queremos que la calle y su altura aparezcan juntas podemos utilizar el parámetro maxsplit de split ().

```
print([re.split(":? ", entrada, 3) for entrada in entradas])
```

Lo cual nos retorna:

```
[['Juan', 'Perez', '15433476', 'Rivadavia 3456'], ['Rosita', 'Bonita', '11329876', 'Uruguay 454'], ['Elva', 'Luarte', '11098567', 'Concordia 321'], ['']]
```

Bibliografía utilizada y sugerida

Libros

Programming Python 5th Edition – Mark Lutz – O'Reilly 2013

Programming Python 4th Edition – Mark Lutz – O'Reilly 2011

Manual online

https://docs.python.org/3.7/tutorial/

https://docs.python.org/3.7/library/index.html

https://www.crummy.com/software/BeautifulSoup/bs4/doc/

https://www.crummy.com/software/BeautifulSoup/bs4/doc/