

"Simple es mejor que complejo" (Tim Peters)

Tutorial de Python Tutorial de IPython Tutorial de EasyGUI Tutorial de Tkinter Tutorial de JupyterLab

domingo, 9 de febrero de 2014

## Expresiones Regulares. Módulo re

El módulo **re** cuenta con funciones para trabajar con expresiones regulares y cadenas.

### La función match()

La función **match** comprueba si una expresión regular tiene coincidencias con el comienzo de una cadena de texto. Se basa en el siguiente formato:

**match(expresiónregular, cadena, [flag])**

Valores de flag:

flag re.IGNORECASE: No se hará diferencia entre mayúsculas y minúsculas  
flag re.VERBOSE: Los comentarios y espacios son ignorados (en la expresión).

```
import re
cadena1 = 'casa'
cadena2 = 'casas'
cadena3 = 'pasa'

if re.match(cadena1, cadena2):
    print('cadena1 y cadena2 son coincidentes')
else:
    print('cadena1 y cadena2 no son coincidentes')

if re.match(cadena1, cadena3):
    print('cadena1 y cadena3 son coincidentes')
else:
    print('cadena1 y cadena3 no son coincidentes')
```

### Comodines

Con **match** el punto "." actúa como un comodín para un solo carácter y representa a cualquier carácter, excepto \n

```
import re
if re.match('.asa', cadena1) and re.match('.asa', cadena3):
    print('cadena1 y cadena3 terminan en .asa')
else:
    print('cadena1 y cadena3 no terminan en .asa')
```

### Carácter especial

Cualquier **carácter especial** se escribirá detrás de una barra invertida ". Por ejemplo, para expresar el carácter punto y no el comodín lo indicaremos escribiendo \.

```
import re
extension = '.jpg'
if re.match(extension, '.jpg') != None:
    print('El archivo es una imagen jpg')
```

### Alternativas

La **barra vertical** "|" expresa distintas alternativas que podrán darse para que se cumpla la expresión.

```
import re
extensiones = ['jpg', 'png', 'gif', 'mp3', 'doc']

for tipoarchivo in extensiones:
    if re.match('jpg|png|gif|bmp', tipoarchivo):
```

Buscar

Buscar

### Python para impacientes

[Índice del Tutorial de Python 3](#)  
[Índice del Tutorial de IPython](#)  
[Índice del Tutorial de EasyGUI](#)  
[Índice del Tutorial de Tkinter](#)  
[Índice del Tutorial de JupyterLab](#)

### Anexos

[Guía urgente de MySQL](#)  
[Guía rápida de SQLite3](#)

### Entradas + populares

[Tkinter: interfaces gráficas en Python](#)

Introducción Con Python hay muchas posibilidades para programar una interfaz gráfica de usuario ( GUI ) pero Tkinter es fácil de...

[Operaciones con fechas y horas. Calendarios](#)

Los módulos datetime y calendar amplían las posibilidades del módulo time que provee funciones para manipular expresiones de ti...

[Gráficos en IPython](#)

Unos de los motivos que inspiraron el desarrollo de IPython fue contar con una herramienta que uniera la posibilidad de realizar cálcu...

[Tkinter: Diseñando ventanas gráficas](#)

Introducción Para definir el modo en que deben colocarse los widgets (controles) dentro de una ventana se utilizan los gestores...

[Dar color a las salidas en la consola](#)

En Python para dar color a las salidas en la consola (o en la terminal de texto) existen varias posibilidades. Hay un método basado ...

[Threading: programación con hilos \(I\)](#)

En programación, la técnica que permite que una aplicación ejecute simultáneamente varias operaciones en el mismo espacio de proceso se...

[Instalación de Python, paso a paso](#)

Instalación de Python 3.6 A finales de 2016 se produjo el lanzamiento de Python 3.6 . El propósito de esta entrada es mostrar, pas...

[Cálculo estadístico](#)

El módulo statistics agrupa un conjunto de funciones para cálculo estadístico. Las funciones están organizadas en dos grupos: las...

[Operaciones con archivos CSV](#)

Un archivo CSV (de Valores Separados por Comas) es un tipo de documento que representa los datos de forma parecida a una tabla, es d...

[Tkinter: Tipos de ventanas](#)

Ventanas de aplicación y de diálogos En la entrada anterior tratamos los distintos gestores de geometría que se utilizan para dis...

```

        print('La extensión ', tipoarchivo, 'se corresponde con una ima
gen')
    else:
        print('La extensión ', tipoarchivo, 'no se corresponde con una
imagen')

```

## Grupos aislados

Los **paréntesis** “()” permiten aislar un grupo de caracteres que pueden ser distintos.

```

import re

palabras = ['careta', 'carpeta', 'colita', 'cateta', 'cocreta', 'caleta',
            'caseta']
for termino in palabras:
    if re.match('ca(...|...)ta', termino):
        print(termino) # careta , carpeta, cateta, caleta, caseta

maspalabras = ['masa', 'mata', 'mar', 'mana', 'cama', 'mareas']
for termino in maspalabras:
    if re.match('ma(s|m|n)a', termino):
        print(termino) # masa, mana

```

## Rangos

Los **corchetes** “[]” se emplean para expresar rangos de números, alfabéticos y de otros caracteres especiales.

```

import re
codigos = ['se1', 'se9', 'ma2', 'se:', 'se.',
           'se2', 'hu2', 'se3', 'sea', 'sec']

for elemento in codigos:
    if re.match('se[0-5]', elemento): # el 3er carácter puede ser nº de 0-5
        print(elemento)

for elemento in codigos:
    if re.match('se[0-5a-z]', elemento): # nº de 0 a 5 y letra de a a z
        print(elemento)

for elemento in codigos:
    if re.match('se[.:]', elemento): # el tercer carácter puede ser . ó :
        print(elemento)

for elemento in codigos:
    if re.match('se[^0-2]', elemento): # debe comenzar por nº de 0 a 2
        print(elemento)

```

## Caracteres predefinidos

**\d** Cualquier carácter que sea dígito  
**\D** Cualquier carácter que no sea dígito  
**\w** Cualquier carácter alfanumérico  
**\W** Cualquier carácter no alfanumérico  
**\s** Espacio en blanco  
**\S** Cualquier carácter que no sea espacio

```

import re
for elemento in codigos:
    if re.match('se\d', elemento): # el 3er carácter debe ser número
        print(elemento)

```

## Caracteres que permiten repeticiones

**+** El carácter de la izquierda aparecerá una o varias veces  
**\*** El carácter de la izquierda aparecerá cero o más veces  
**?** El carácter de la izquierda aparecerá cero o una vez  
**{}** Indica el número de veces que debe aparecer el carácter de la izquierda:

**{3}** 3 veces; **{1,4}** de 1 a 4; **{,3}** de 0 a 3; **{2,}** dos o más veces

```

import re
codigos = ['aaa111', 'aab11', 'aaa1111', 'aaz1', 'aaa']

```

## Archivo

Archivo 

## Twitter



## Sitios

- [ActivePython](#)
- [Anaconda](#)
- [Bpython](#)
- [Django](#)
- [Flask](#)
- [Ipython](#)
- [IronPython](#)
- [Matplotlib](#)
- [MicroPython](#)
- [Numpy](#)
- [Pandas](#)
- [Pillow](#)
- [PortablePython](#)
- [PyBrain](#)
- [PyCharm](#)
- [PyDev](#)
- [PyGame](#)
- [Pypi](#)
- [PyPy](#)
- [Pyramid](#)
- [Python.org](#)
- [PyTorch](#)
- [SciPy.org](#)
- [Spyder](#)
- [Tensorflow](#)
- [TurboGears](#)

```
for elemento in codigos:
    if re.match('aa[a-z]1{2,}', elemento):
        print(elemento) # aaal11 , aabl1, aaal111

for elemento in codigos:
    if re.match('a+1+', elemento):
        print(elemento) # aaal11 , aaal111
```

## El objeto `mo` y el método `group()`

El método **group** del objeto **mo** devuelve la cadena encontrada o produce excepción

```
import re
mo = re.match('ftp://.+\.com', 'ftp://ovh.com')
print(mo.group()) # ftp://ovh.com
```

Con los paréntesis acotamos los grupos:

```
import re
mo = re.match('ftp://(.+)\.com', 'ftp://ovh.com')
print(mo.group(0)) # ftp://ovh.com
print(mo.group(1)) # ovh.
print(mo.groups()) # ('ovh.',).
```

## La función `search()`

La función **search** es como **match** pero busca coincidencias de un patrón en una cadena de texto y dichas coincidencias pueden aparecer en cualquier lugar. Su formato es: **search(patrón, cadena, [flag])**

```
import re
palabras = ['paniaguado', 'agüita', 'aguador',
            'paraguas', 'agua']

for elemento in palabras:
    if re.search('agua', elemento):
        print(elemento) # paniaguado, aguador, paraguas, agua
```

## Coincidencias al comienzo y al final

Busca una subcadena al **^ COMIENZO** o al **\$ FINAL** de una cadena:

```
import re
lista_url = ['http://www.aaa.es',
             'ftp://www.aaa.es',
             'http://www.bbb.es']

for elemento in lista_url:
    if re.search('^ftp://', elemento):
        print(elemento) # ftp://www.aaa.es

lista_dom = ['.com', '.es']
for elemento in lista_dom:
    if re.search('es$', elemento):
        print(elemento) # .es
```

## Métodos `start()` y `end()`

El método **start()** devuelve la posición inicial y el método **end()** la final, si la subcadena está en la cadena.

```
import re
mol = re.search('agua', 'paraguas')
print(mol.start()) # devuelve 3
print(mol.end()) # devuelve 7
```

## La función `findall()`

La función **findall()** devuelve una lista con las subcadenas que cumplen el patrón en una cadena. El formato que utiliza es: **findall(patrón, cadena, [flag])**

```
cadena = 'tengo una yama que yama se llama'
lista = re.findall('..ama', cadena)
print(lista) # muestra: [' yama', ' yama', 'llama']
```

## La función `finditer()`

La función **finditer()** permite usar un iterador para recorrer las subcadenas que cumplen el patrón. El resultado son tuplas con las posiciones de las subcadenas.

```
cadena = 'tengo una yama que yama se llama'
iterador = re.finditer('ama', cadena)
for encontrado in iterador:
    print(encontrado.span()) # (11, 14) , (20, 23) , (29, 32)
```

### La función compile()

La función **compile()** se utiliza para compilar una expresión regular, devolviendo un objeto especial llamado *RegexObject*. La compilación es un paso previo que conlleva la evaluación del patrón que indiquemos en la función; que después utilizaremos con las funciones **split()**, **sub()**, **subn()** y otras.

Es importante señalar que la mayoría de las operaciones con expresiones regulares que están disponibles como métodos compilados a nivel de módulo, están también como funciones, con algunas diferencias en sus parámetros. Las funciones son atajos que no requieren el paso de la compilación.

### La función sub() con compilación

La función **sub()** busca y sustituye cadenas usando el siguiente formato:

**sub(cadenaparasustituir, cadenadondesebusca, [count=número])**

En el ejemplo siguiente se declara una clave y todos sus caracteres no numéricos se sustituyen por "0":

```
import re
clave = "asdb92z$"

# \D se refiere a cualquier carácter que no es número
patron = re.compile("\D")

# Se sustituyen los caracteres encontrados por "0"
nueva_clave = patron.sub("0", clave)

print(nueva_clave) # 00009200

# Para mostrar el tipo de objeto de "patron":
print(type(patron))

# Otra forma de expresarlo:
nueva_clave = re.compile("\D").sub("0", clave)
```

#### Otros ejemplos

```
import re
oracion = 'la norma es la norma'
patron = re.compile('norma')
print(patron.sub('ley', oracion)) # la ley es la ley

patron = re.compile('la')
print(patron.sub('LA', oracion, count=1))
# LA norma es la norma
```

### La función subn() con compilación

La función **subn()** es como **sub()** pero devuelve una tupla con dos valores: el primero contiene la cadena resultado después de aplicar las sustituciones y en el segundo el número de sustituciones realizadas.

En el siguiente ejemplo en vez de utilizar el patrón "\D" se emplea "\d" que se refiere a todos los caracteres numéricos. Dicho patrón se utilizará para sustituir todos los caracteres numéricos por el carácter "x":

```
clave = "asdb92z$"
patron = re.compile("\d")
tupla_resultado = patron.subn("x", clave)
print(tupla_resultado[0]) # asdbxxz$
print(tupla_resultado[1]) # 2
```

### La función split() con compilación

La función **split()** divide una cadena en subcadenas: **split(cadena, [maxsplit=0])**

```
import re
meses = 'ene+feb+mar+abr+may+jun'

patron = re.compile('\+')
print(patron.split(meses))
# ['ene', 'feb', 'mar', 'abr', 'may', 'jun']

patron = re.compile('\+')
print(patron.split(meses, maxsplit=1))
# ['ene', 'feb+mar+abr+may+jun']
```

[Ir al índice del tutorial de Python](#)

Publicado por Pherkad en 6:38



Etiquetas: [Python3](#)

## 2 comentarios:

**Anónimo dijo...**

Demasiado impaciente!!

Falta explicar el re.compile ...que apareció de la nada, sin decir: "agua va!"

Un abrazo

[16 de enero de 2015, 17:21](#)



**Pherkad dijo...**

je, je, je, muy gráfico lo de "agua va!". Y es cierto: aquí faltaba alguna aclaración previa, que he incluido ya, junto a algún ejemplo nuevo.

También, he añadido información sobre la función subn() con compilación.

Muchas gracias por el comentario!

[17 de enero de 2015, 7:31](#)

[Publicar un comentario](#)

[Entrada más reciente](#)

[Página principal](#)

[Entrada antigua](#)

Suscribirse a: [Enviar comentarios \(Atom\)](#)