

Expresiones regulares

Una expresión regular es un patrón que describe una colección de cadenas de caracteres.

Por ejemplo, el patrón `a.u` representa todas las cadenas de caracteres que empiezan por la letra `a`, luego un carácter cualquiera (representado por el símbolo `.`) y luego la letra `u`. Más concretamente, el patrón `a.u` representa las cadenas `abu`, `agu` y `a-u`, pero no las cadenas `Abu` o `ave`.

Otro ejemplo: el patrón `"ab*"` representa las cadenas de caracteres que empiecen por la letra `a` seguida por cero o más caracteres `b`.

Y otro: el patrón `"qui(jote|jano|mera)"` representa las cadenas de caracteres que empiecen por `qui`, seguidas `jote`, por `jano` o por `mera`. Concretamente, las cadenas `"quijote"`, por `"quijano"` o por `quimera`.

La función `match()`

La función `match()` averigua si una cadena de caracteres contiene un patrón, *en su comienzo*.

Por ejemplo, como el símbolo `"."` representa exactamente un carácter cualquiera, el patrón `"a.u"` encaja al comienzo de las cadenas `"abuelo"`, `"avuelo"`, `"avutarda"` y `"aguja"`, pero no de las cadenas `"Abuelo"` y `"aaaaa"`.

In [1]:



```
import re

patron = re.compile("a.u")

cadena = "abuelo"
if re.match(patron, cadena):
    print("El patrón encaja en la cadena " + cadena)
else:
    print("El patrón NO encaja en la cadena " + cadena)

cadena = "Abuelo"
if re.match(patron, cadena):
    print("El patrón encaja en la cadena " + cadena)
else:
    print("El patrón NO encaja en la cadena " + cadena)

for cadena in ["abuelo", "Abuelo", "avuelo", "avutarda", "aguja", "aaaaa"]:
    if re.match(patron, cadena):
        print(cadena)
```

```
El patrón encaja en la cadena abuelo
El patrón NO encaja en la cadena Abuelo
abuelo
avuelo
avutarda
aguja
```

Otro comodín: "*"

Otro comodín es el símbolo "*", que representa exactamente la repetición "ninguna o más veces", del carácter precedente. Así por ejemplo, el patrón "ab*o" representa el comienzo de las cadenas "abono" y "abbbbbonar" o "ao", pero no de las cadenas "abuelo" o "aaaaa".

In [2]:

```
patron = re.compile("ab*o")

for cadena in ["abuelo", "Abuelo", "abono", "abbbbbonar", \
               "abuela", "abuelitos", "aaaaa", "ao"]:
    if re.match(patron, cadena):
        print(cadena)
```

```
abono
abbbbbonar
ao
```

Algunos patrones básicos

El símbolo "." en el patrón representa un carácter cualquiera, y "*", ninguna o más repeticiones del símbolo precedente. En los patrones se usan otros muchos símbolos. Veamos algunos ejemplos para empezar:

Algunos códigos especiales nos permiten identificar dígitos, caracteres no dígitos...

Patrón	Significado
"a*"	El carácter a, ninguna o más veces
"a+"	El carácter a, una o más veces
"[a-z]"	Una letra de la "a" a la "z"
"[1-9]+"	Un dígito entre uno y nueve, una o más veces
"mi(\\.\\.\\.o)"	Empieza por "mi", luego uno o dos caracteres y luego una "o": mito, mico, mirlo, miedo
"^The.*Spain\$"	Empieza por "The" y termina con "Spain"

La función search() :

La función search() recorre una cadena y busca en ella algún fragmento que encaje con el patrón dado, y da el fragmento posible en caso de encontrarlo:

In [3]:



```
patron = re.compile("c...")

encaje = re.search(patron, "En un lugar...")
print(encaje)

encaje = re.search(patron, "... de la Mancha, de cuyo nombre no quiero acordarme...")
print(encaje)
print(encaje.start())
print(encaje.end())
```

None

<re.Match object; span=(13, 17), match='cha,'>

13

17

La función findall():

Podríamos desear ver todos los encajes, y no sólo el primero: nuestra función es ahora findall() :

In [4]:



```
cadena = """En un lugar de la Mancha de cuyo nombre\
no quiero acordarme, había un hidalgo, de los de\
lanza en astillero, rocín flaco y galgo corredor..."""

print(re.findall("c...", cadena))

print(re.findall(". l..", cadena))
```

['cha ', 'cuyo', 'cord', 'cín ', 'co y', 'corr']

['n lug', 'e la ', 'e los']

La función split():

Separa una cadena según un patrón:

In [5]:



```
cadena = """En un lugar de la Mancha de cuyo nombre\
no quiero acordarme, había un hidalgo, de los de\
lanza en astillero, rocín flaco y galgo corredor..."""

separada = re.split("c...", cadena)
print(separada)
```

```
['En un lugar de la Man', 'de ', ' nombreno quiero a', 'arme, había un hidal
go, de los delanza en astillero, ro', 'fla', ' galgo ', 'edor...']
```

La función sub():

Sustituye las apariciones de un patrón por otra cosa, en una cadena de caracteres:

In [6]:



```
cadena = """En un lugar de la Mancha de cuyo nombre\
no quiero acordarme, había un hidalgo, de los de\
lanza en astillero, rocín flaco y galgo corredor..."""

separada = re.sub("c...", "----", cadena)
print(separada)
```

En un lugar de la Man----de ---- nombreno quiero a----arme, había un hidalgo, de los delanza en astillero, ro----fla---- galgo ----edor...

La función group() :

Dentro de un fragmento que encaja con un patrón, podemos distinguir partes. La función group permite definir estas partes y localizarlas por separado:

In [7]:



```
cadena = """En un lugar de la Mancha de cuyo nombre\
no quiero acordarme, había un hidalgo, de los de\
lanza en astillero, rocín flaco y galgo corredor..."""

patron = re.compile("c....")
encaje = re.search(patron, cadena)
print(encaje.group())

patron = re.compile("c((..)(..))")
encaje = re.search(patron, cadena)
print(encaje.group(0))
print(encaje.group(1))
print(encaje.group(2))
print(encaje.group(3))
```

```
cha d
cha d
ha d
ha
d
```

Patrones

En los ejemplos anteriores hemos visto los patrones siguientes:

- El patron "ab*", que representa las cadenas de caracteres que empiezan por la letra `a` seguida de cero o más caracteres "b"
- El patrón ". l..", que representa las cadenas de cinco caracteres: uno cualquiera, un espacio, la letra "l" y dos caracteres cualesquiera más.

Pero hay otros patrones posibles, que se pueden definir con las siguientes convenciones. Vamos a verlos mediante ejemplos:

In [8]:



```
patrones = ['.ab*',      # un carácter, el carácter "a" seguido por cero o más caracteres
            '.ab+',      # un carácter, el carácter "a" seguido por uno o más caracteres
            '.ab?',      # un carácter, el carácter "a" seguido por cero o un carácter "b"
            '.ab{2}',     # un carácter, el carácter "a" seguido por dos caracteres "b".
            '.ab{2,4}',   # un carácter, el carácter "a" seguido por 2, 3 o 4 caracteres "b"
            '.[ab].',     # "[ab]" es el carácter "a" o el carácter "b".
            '.[ab]+',     # un carácter seguido de uno o más caracteres "a" o "b"
            ]

frase = "0a 1b 2ab 3aba 4abc 5aaaaaaaa 6abbab 7abbbbb 6abababababab"

for p in patrones:
    print("Búsqueda con el patrón '" + p + "'")
    print(re.findall(p, frase))
```

Búsqueda con el patrón `'.ab*'`

```
['0a', '2ab', '3ab', '4ab', '5a', 'aa', 'aa', 'aa', '6abb', '7abbbbb', '6a
b', 'bab', 'bab']
```

Búsqueda con el patrón `'.ab+'`

```
['2ab', '3ab', '4ab', '6abb', '7abbbbb', '6ab', 'bab', 'bab']
```

Búsqueda con el patrón `'.ab?'`

```
['0a', '2ab', '3ab', '4ab', '5a', 'aa', 'aa', 'aa', '6ab', 'bab', '7ab', '6a
b', 'bab', 'bab']
```

Búsqueda con el patrón `'.ab{2}'`

```
['6abb', '7abb']
```

Búsqueda con el patrón `'.ab{2,4}'`

```
['6abb', '7abbbbb']
```

Búsqueda con el patrón `'.[ab].'`

```
['0a ', '1b ', '2ab', '3ab', '4ab', '5aa', 'aaa', 'aa ', '6ab', 'bab', '7a
b', 'bbb', '6ab', 'aba', 'bab', 'aba']
```

Búsqueda con el patrón `'.[ab]+'`

```
['0a', '1b', '2ab', '3aba', '4ab', '5aaaaaaaa', '6abbab', '7abbbbb', '6ababab
ababab']
```

Secuencias de escape

Algunos códigos especiales nos permiten identificar dígitos, caracteres no dígitos...

Código	Significado
<code>"\d"</code>	un dígito
<code>"\D"</code>	un carácter no dígito
<code>"\s"</code>	espacio en blanco o tabulador o nueva línea, etc.)
<code>"\S"</code>	no espacio en blanco o...
<code>"\w"</code>	carácter alfaumérico
<code>"\W"</code>	no carácter alfaumérico

In [9]:

```
patrones = ["\\d+", # Secuencia de dígitos
            "\\D+", # Secuencia de no dígitos
            "\\s+", # Secuencia de espacios en blanco
            "\\S+", # Secuencia de no espacios en blanco
            "\\w+", # Secuencia de caracteres alfanuméricos
            "\\W+", # Secuencia de no caracteres alfanuméricos
            ]

frase = "¡El número del anticristo es 666, el número de la bestia!"

for p in patrones:
    print("Búsqueda con el patrón " + p)
    print(re.findall(p, frase))
```

```
Búsqueda con el patrón \d+  
['666']  
  
Búsqueda con el patrón \D+  
['¡El número del anticristo es ', ' ', el número de la bestia!']  
  
Búsqueda con el patrón \s+  
[' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']  
  
Búsqueda con el patrón \S+  
['¡El', 'número', 'del', 'anticristo', 'es', '666,', 'el', 'número', 'de',  
'la', 'bestia!']  
  
Búsqueda con el patrón \w+  
['El', 'número', 'del', 'anticristo', 'es', '666', 'el', 'número', 'de', 'l  
a', 'bestia']  
  
Búsqueda con el patrón \W+  
['!', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '!']
```

In [10]:

```
patrones = ["[^!.? ]+", # "Un carácter que no es ni "!", ni ".", ni "?" ni " ".
            "[a-z]",      # rango de caracteres
            "[a-zA-Z]",    # una minúscula o una mayúscula
            "[A-Z][a-z]", # una minúscula seguida de una mayúscula
            ]

frase = "¡Qué lindos ojos! ¿Puedes decirme tu nombre?"

for p in patrones:
    print("Búsqueda con el patrón " + p)
    print(re.findall(p, frase))
```

```
Búsqueda con el patrón [^!.? ]+
['¡Qué', 'lindos', 'ojos', '¿Puedes', 'decirme', 'tu', 'nombre']
Búsqueda con el patrón [a-z]
['u', 'l', 'i', 'n', 'd', 'o', 's', 'o', 'j', 'o', 's', 'u', 'e', 'd', 'e', 's', 'd', 'e', 'c', 'i', 'r', 'm', 'e', 't', 'u', 'n', 'o', 'm', 'b', 'r', 'e']
Búsqueda con el patrón [a-zA-Z]
['Q', 'u', 'l', 'i', 'n', 'd', 'o', 's', 'o', 'j', 'o', 's', 'P', 'u', 'e', 'd', 'e', 's', 'd', 'e', 'c', 'i', 'r', 'm', 'e', 't', 'u', 'n', 'o', 'm', 'b', 'r', 'e']
Búsqueda con el patrón [A-Z][a-z]
['Ou', 'Pu']
```

Ejemplos adicionales

In [11]:

```
# Patrón para identificar una dirección de email:
patron_email = r"([a-zA-Z0-9_+-.]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-]+\.)+"
frase = "Mi email es cpareja@ucm.es, no c.pareja@sip.ucm.es ni c_pareja@SIP.ucm.es.es.es"
print(re.findall(patron_email, frase))
```

```
['cpareja@ucm.es', 'c.pareja@sip.ucm.es', 'c_pareja@SIP.ucm.es.es.es']
```

In [12]:

```
# Patrón para identificar una fecha:
fecha_re = re.compile('\d{2}/\d{2}/\d{4}')

linea = '[26/11/1962 00:01:35] <font color="#00ff00">¡Sorpresa!</font>>'
encaje = fecha_re.search(linea)
print(encaje)
print(encaje.group(0))

linea_sin_fecha = "En tiempos de Ahrun al Rashid..."
encaje = fecha_re.search(linea_sin_fecha)
print(encaje)
```

```
<re.Match object; span=(1, 11), match='26/11/1962'>
26/11/1962
None
```

In [13]:

```
# Patrón para identificar una fecha y una hora:

linea = '[26/11/1962 00:01:35] <font color="#00ff00">¡Sorpresa!</font>>'

# Ahora definimos dos grupos, con los paréntesis:
fecha_con_hora = re.compile('(\d{2}/\d{2}/\d{4}) (\d{2}:\d{2}:\d{2})')

encaje = fecha_con_hora.search(linea)
print(encaje)
print(encaje.group(0))
print(encaje.group(1))
print(encaje.group(2))
```

```
<re.Match object; span=(1, 20), match='26/11/1962 00:01:35'>
26/11/1962 00:01:35
26/11/1962
00:01:35
```

In [14]:



```
# Para reconvertir Las cadenas de caracteres de fechas en fechas:
from datetime import datetime
fecha = datetime.strptime(encaje.group(0), "%d/%m/%Y %H:%M:%S")
fecha

# https://docs.python.org/3.6/Library/datetime.html#strptime-behavior
```

Out[14]:

```
datetime.datetime(1962, 11, 26, 0, 1, 35)
```

Brevísimo comentario final

Las expresiones regulares se pueden manejar en Python mediante la librería "re", que se ha de importar para poder usar las funciones para manejar los patrones y las funciones "match()", "search()", "findall()", etc., con las que manejaremos las expresiones regulares.

Pero el tema en sí es bastante más amplio en el mundo de la Computación, pues involucra conceptos de autómatas finitos deterministas y no deterministas, compiladores, etc., que no consideramos aquí obviamente.

Entre las muchas referencias que pueden consultarse, he aquí dos que me gustan a mí:

- https://www.w3schools.com/python/python_regex.asp
- <https://docs.python.org/3/howto/regex.html>