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

def comprobar_encaje(patron_str, cadena):
    patron = re.compile(patron_str)
    if re.match(patron, cadena):
        # print(f"El patrón '{patron_str}' SÍ encaja en la cadena '{cade
        print(f"'{patron_str}' === '{cadena}'")
    else:
        # print(f"El patrón '{patron_str}' NO encaja en la cadena '{cade
        print(f"'{patron_str}' =/= '{cadena}'")

comprobar_encaje("a.u", "abuelo")
comprobar_encaje("a.u", "Abuelo")
```

```
'a.u' === 'abuelo'
'a.u' =/= 'Abuelo'
```

In [2]:
```
for cad in ["abuelo", "Abuelo", "avuelo", "avutarda", "aguja", "aaaaa"]:
    comprobar_encaje("a.u", cad)
```

```
'a.u' === 'abuelo'
'a.u' =/= 'Abuelo'
'a.u' === 'avuelo'
'a.u' === 'avutarda'
'a.u' === 'aguja'
'a.u' =/= 'aaaaa'
```

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 "abbbbonar" o "ao", pero no de las cadenas "abuelo" o "aaaaa".

In [3]:
```
for cad in ["abuelo", "Abuelo", "abono", "abbbbonar",
            "abuela", "abuelitos", "aaaaa", "ao"]:
    comprobar_encaje("ab*o", cad)
```

```
'ab*o' =/= 'abuelo'
'ab*o' =/= 'Abuelo'
'ab*o' === 'abono'
'ab*o' === 'abbbbonar'
'ab*o' =/= 'abuela'
'ab*o' =/= 'abuelitos'
'ab*o' =/= 'aaaaa'
'ab*o' === '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
"."	Un carácter
"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"

In [4]:
```
cadenas = ["abuelo", "Abuelo", "abono", "abbbbonar",
           "abuela", "abuelitos", "aaaaa", "ao"]

for cad in cadenas:
        comprobar_encaje("a.*b+", cad)

print(".....................................")

for cad in cadenas:
        comprobar_encaje("a(b|v|g).*", cad)

print(".....................................")

for cad in ["En un lugar de la Mancha", "En la Mancha..."]:
        comprobar_encaje("^En.*Mancha$", cad)
```

```
'a.*b+' === 'abuelo'
'a.*b+' =/= 'Abuelo'
'a.*b+' === 'abono'
'a.*b+' === 'abbbbonar'
'a.*b+' === 'abuela'
'a.*b+' === 'abuelitos'
'a.*b+' =/= 'aaaaa'
'a.*b+' =/= 'ao'
.....................................
'a(b|v|g).*' === 'abuelo'
'a(b|v|g).*' =/= 'Abuelo'
'a(b|v|g).*' === 'abono'
'a(b|v|g).*' === 'abbbbonar'
'a(b|v|g).*' === 'abuela'
'a(b|v|g).*' === 'abuelitos'
'a(b|v|g).*' =/= 'aaaaa'
'a(b|v|g).*' =/= 'ao'
.....................................
'^En.*Mancha$' === 'En un lugar de la Mancha'
'^En.*Mancha$' =/= 'En la Mancha...'
```

El método `search()`:

El método `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 [5]:
```
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 a
print(encaje)
print(encaje.start())
print(encaje.end())
```

```
None
<re.Match object; span=(13, 17), match='cha,'>
13
17
```

El encaje con una cadena completa puede realizarse también con el método `re.fullmatch()`, que comprueba si toda la cadena coincide con el patrón, y no sólo una parte de ella.

In [6]:
```
for cad in ["En un lugar de la Mancha", "En la Mancha..."]:
        print(re.fullmatch("En.*Mancha", cad))
```

```
<re.Match object; span=(0, 24), match='En un lugar de la Mancha'>
None
```

La función `findall()`:

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

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..."""

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

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

```
['cha ', 'cuyo', 'cord', 'cín ', 'co y', 'corr']
['n lug', 'e la ', 'e los', 'e lan']
```

Algunos patrones más

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

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

for p in patrones:
    print(f"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',
'6ab', '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', '7a
b', '6ab', 'bab', 'bab']
Búsqueda con el patrón '.ab{2}'
['6abb', '7abb']
Búsqueda con el patrón '.ab{2,4}'
['6abb', '7abbbb']
Búsqueda con el patrón '.[ab].'
['0a ', '1b ', '2ab', '3ab', '4ab', '5aa', 'aaa', 'aa ', '6ab', 'bab',
'7ab', 'bbb', '6ab', 'aba', 'bab', 'aba']
Búsqueda con el patrón '.[ab]+'
['0a', '1b', '2ab', '3aba', '4ab', '5aaaaaaa', '6abbab', '7abbbbb', '6a
bababababab']
```

Veamos algunos patrones más:

In [9]:
```
patrones = ["[^!.? ]+",   # Un carácter que no es "!", 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 tienes! ¿Puedes decirme tu nombre?"

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

```
Búsqueda con el patrón '[^!.? ]+':
['¡Qué', 'lindos', 'ojos', 'tienes', '¿Puedes', 'decirme', 'tu', 'nombr
e']
Búsqueda con el patrón '[a-z]':
['u', 'l', 'i', 'n', 'd', 'o', 's', 'o', 'j', 'o', 's', 't', 'i', 'e',
'n', 'e', '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', 't', 'i',
'e', 'n', 'e', '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]':
['Qu', 'Pu']
```

In [10]:
```
# Ejemplo: un número entero, con signo o sin él:

entero = re.compile(r'[\+\-]?[0-9]+')
cantidades = re.findall(entero, "Tengo 123 euros, en el banco, -150€ y g

print(cantidades)
print(sum([int(c) for c in cantidades]))
```

```
['123', '-150', '+347']
320
```

El método `split()`:

Separa una cadena según un patrón:

In [11]:
```
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)

print(".....................................")

separadores_de_frase = "[.;:] " # uno de esos caracteres + un espacio en
frases = re.split(separadores_de_frase,
                  "Estoy de acuerdo. Pero no del todo; otro día lo discu
print(frases)
```

```
['En un lugar de la Man', 'de ', ' nombre no quiero a', 'arme, había un
hidalgo, de los de lanza en astillero, ro', 'fla', ' galgo ', 'edo
r...']
.....................................
['Estoy de acuerdo', 'Pero no del todo', 'otro día lo discutimos', 'aho
ra no puedo.']
```

El método `sub()`:

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

In [12]:
```
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 ---- nombre no quiero a----arme, había un h
idalgo, de los de lanza en astillero, ro----fla---- galgo ----edor...
```

El método `group()`:

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

In [13]:
```
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
```

In [14]:
```
# Buscamos un número entero en una cadena:

patr_ent = re.compile("[^0-9]*([0-9]+)[^0-9]")
cadena = """[...] ejemplares únicos en la \
Hispanic Society (Fadrique de Basilea, Burgos, 1499 pero, \
en realidad, 1500-1502), la Sociedad Bodmeriana \
(Pedro Hagenbach, Toledo, 1500) y en la Biblioteca Nacional de Francia \
(Estanislao Polono, Sevilla, 1501) [...]"""

encaje = re.search(patr_ent, cadena)
print(encaje.group(0))
print(encaje.group(1))

# Buscamos ahora todos los enteros:

print(re.findall(patr_ent, cadena))
```

```
[...] ejemplares únicos en la Hispanic Society (Fadrique de Basilea, Bu
rgos, 1499
1499
['1499', '1500', '1502', '1500', '1501']
```

El método `finditer()`

... para buscar en una cadena varios encajes agrupados:

In [15]:
```
# Buscamos un número entero y luego un nombre propio en una cadena:

patr_ent_np = re.compile("[^0-9]*([0-9]+)[^0-9][^A-Z]*([A-Z][a-z]*)")
encajes = re.finditer(patr_ent_np, cadena)
for enc in encajes:
    print("-> ", enc.group(0))
    print("-> ", enc.group(1))
    print("-> ", enc.group(2))
    print()
```

```
->  [...] ejemplares únicos en la Hispanic Society (Fadrique de Basile
a, Burgos, 1499 pero, en realidad, 1500-1502), la Sociedad
->  1499
->  Sociedad

->   Bodmeriana (Pedro Hagenbach, Toledo, 1500) y en la Biblioteca
->  1500
->  Biblioteca
```

Secuencias de escape

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

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

In [16]:

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

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

for p in patrones:
    print(f"'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', 'd
e', 'la', 'bestia!']
'Búsqueda con el patrón '\w+'
['El', 'número', 'del', 'anticristo', 'es', '666', 'el', 'número', 'd
e', 'la', 'bestia']
'Búsqueda con el patrón '\W+'
['¡', ' ', ' ', ' ', ' ', ' ', ', ', ' ', ' ', ' ', ' ', '!']
```

Ejemplos adicionales

In [17]:

```
# 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 [18]:
```
# Patrón para identificar una dirección de email:

dir_email = r'[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+(?:\.[a-zA-Z0-9-]+)+'
dir_email = r'[\w_.+-]+@[\w_.+-]+(?:\.[\w_.+-]+)+'

trozo = "[\w_.+-]+"
dir_email = f'{trozo}@{trozo}(?:\.{trozo})+'

frase = "Mi email: cpareja@ucm.es. No c.par@sip.ucm.es ni c-pa+reja@SIP.
print(re.findall(dir_email, frase))
```

```
['cpareja@ucm.es.', 'c.par@sip.ucm.es', 'c-pa+reja@SIP.ucm.es.es.es']
```

In [19]:
```
# 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 [20]:
```
# 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#strftime-strptime-be
```

Out[20]:
```
datetime.datetime(1962, 11, 26, 0, 1, 35)
```

Comentario final

En Computación, el tema de las expresiones regulares es más amplio de lo mostrado aquí, pues involucra conceptos de autómatas finitos deterministas y no deterministas, compiladores, etc., temas 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

Finalmente, os dejo el siguiente enlace, que permite probar y depurar expresiones regulares:

```
https://regex101.com/
```