

expresiones regulares...

...en Python

expresiones regulares

- Una "Expresión Regular", "RegEx", es una patrón que describe una cantidad de texto.

expresiones regulares

Un texto cualquiera es una expresión regular:

ej.

"Cacho" va a dar coincidencia ("match") en todo texto que contenga ese patrón:

"Cacho" el mejor" → match

Un **cacho** de Cultura" → ~~match~~ (cacho está con minúsculas)

expresiones regulares

Se puede usar `|` para especificar “un texto u otro” (`|` → OR)

ej.:

Cacho | **cacho** va a dar coincidencia (“match”) en todo texto que contenga ese **Cacho** o **cacho**:

"**Cacho** el mejor" → match

"Un **cacho** de Cultura" → match

expresiones regulares

- Para especificar un patrón, se pueden usar caracteres especiales (metacaracteres) que permiten mayor flexibilidad

[] especifica una clase de caracteres; indica que **en ese lugar** del patrón puede haber cualquier letra de la clase.

[A-E] → cualquier caracter, de A a E

[a-z0-9] → cualquier letra minúscula o número

[Uu] → U mayúscula o minúscula

expresiones regulares

Ej.

`"[cC]acho"` coincide con "Cacho" o "cacho"

`"[A-Z][A-Z][A-Z]"` coincide con "ABC" y "ZZT" pero no con "VF4"

`"[mby]ate"` coincide con "mate", "bate" o "yate"

`"img[_-][1-9]"` coincide con "img_2", "img-8" o "img_1"

expresiones regulares

- Para indicar repetición, se usa { }

{a} el grupo o caracter anterior se repite a veces

{a,b} el grupo o caracter anterior se repite entre a y b veces

{,b} el grupo o caracter anterior se repite entre 0 y b veces

{a,} el grupo o caracter anterior se repite a o más veces

expresiones regulares

Ej.

`"[A-Z]{3}"` coincide con "MNO"

`"[A-Z]{2,4}"` coincide con "AY", "MNU", "WXYZ"

`"doc_[0-9]{2,}"` coincide con "doc_01", "doc_145" o "doc_8545678"

`"img[1-9]{,3}"` coincide con "img", "img5", "img911"

`19[0-9]{2}` coincide con cualquier año del siglo XX

expresiones regulares

- También se puede usar:
 - + (1 o más veces) - es como usar $\{1, \}$
 - * (0 o más veces) - es como usar $\{0, \}$
 - ? (0 o 1 vez) - es como usar $\{0, 1\}$

expresiones regulares

Ej.

"g[o]+1" coincide con "gol" , "goool", etc.

"Ouch!*" coincide con "Ouch", "Ouch!!!!!"

"Dra?" coincide con "Dr", "Dra"

expresiones regulares

- Para "anclar" un patrón al inicio y al fin del string se usa `^` y `$`

`"^Sra{,1}"` coincide si el string comienza con Sr o Sra

`"-fin-$"` coincide si el string termina con -fin-

`"^[A-Z]{3}[0-9]{3}$"` coincide si el string está formado por 3 letras mayúsculas seguidas de 3 números (3 letras al inicio, 3 letras al final)

`"^[A-Za-z]{4}$"` coincide cualquier palabra de 4 letras

expresiones regulares

- Para "negar" se usa `^` dentro de una clase o grupo: agrupamos con ()

`"^[^a-z][a-z]*"` no comienza con minúsculas: El primer `^` es un ancla (anchor) al inicio del string. El `^` dentro de los corchetes es el que "niega"

`"[^aA0-9]na"` coincide con una y ona pero no con Ana, ana o 1na

expresiones regulares

- `\d` dígitos equivale a `[0-9]`
- `\w` caracteres equivale a `[A-Za-z0-9_]`
- `\s` "espacios" equivale a `[ \t\r\n\f]` - espacio, tab, carriage return,
new line, line feed-
- `.` cualquier caracter excepto `\n`

Si usamos las letras en mayúscula, negamos la clase: `\D` equivale a `[^0-9]`,
`\W` equivale a `[^A-Za-z0-9_]`, `\S` equivale a `[^\t\r\n\f]`

expresiones regulares

Ej.

`19\d{2}` o `19[0-9]{2}` coincide con cualquier año del siglo XX

`\D{3}` coincide con tres caracteres que **no** sean números

expresiones regulares

- Para indicar que el patrón empieza y/o termina en una transición entre `\w` y `\W` (al inicio o fin de una "palabra") se usa `\b`

`"\b\d{3}"` coincide con 123toto y con 789, no con toto456

`"\d{3}\b"` coincide con toto456 y con 789, no con 123toto

`"\b\d{3}\b"` coincide con 789, no con toto456 ni con 123toto

RegEx en Python

expresiones regulares en Python

- **re** es el módulo de la biblioteca estándar de Python para trabajar con RegEx

```
import re
```

expresiones regulares en Python

- Algunas funciones:

```
re.match(pattern, string)
```

```
re.search(pattern, string, flags=0)
```

```
re.findall(pattern, string)
```

donde:

`pattern` es la expresión regular, y

`string` es el texto donde se buscan coincidencias con la regex

re.search y re.match

- `re.search()` y `re.match()` devuelven un objeto clase `Match` si el patrón se encuentra en el texto
- Si no hay coincidencia, devuelven `None`
- Sólo devuelven la primer coincidencia
- `match()` solo busca **al inicio** del string

re.search y re.match

```
import re

texto = 'Hoy es 5 del 5'

regex = '5'

if re.search(regex, texto):

    print('Hay un 5')

else:

    print('Sin cincos')
```

Hay un cinco

```
import re

texto = 'Hoy es 5 del 5'

regex = '5'

if re.match(regex, texto):

    print('Hay un 5')

else:

    print('Sin cincos')
```

Sin cincos

re.search y re.match

```
import re

texto = 'Hoy es 5 del 5'

regex = '5'

mtch = re.search(regex, texto):

if mtch:

    print('Hay un 5')

else:

    print('Sin cincos')
```

- mtch es un objeto clase Match

```
>>> print(mtch.end())
```

8

```
>>> print(mtch.start())
```

7

```
>>> print(mtch.span())
```

(7, 8)

re.findall

- **re.findall()** devuelve una lista con todas las coincidencias
- si no hay coincidencias, devuelve una lista vacía

```
import re
```

```
texto = 'Hoy es 5 del 5'
```

```
regex = '5'
```

```
print(re.findall(regex, texto))
```

```
['5', '5']
```

```
import re
```

```
texto = '¿En 1989 o en 1984?'
```

```
regex = '19\\d{2}'
```

```
print(re.findall(regex, texto))
```

```
['1989', '1984']
```

rawstrings en regex

- En el último ejemplo de `findall()` usamos como patrón:

`'19\\d{2}'` → usamos `\\` para indicar una `\`

- Para simplificar la sintáxis se puede usar rawstrings anteponiendo `r` al string:

`r'19\d{2}'`

- Ej. `"\\d{3}"` vs `r'\\d{3}'` para tres números entre `\` → `\123\`

re.split

- `re.split()` devuelve una lista con el string original separado en cada coincidencia

```
import re

texto = r'uno<br>dos<br>tres'

rx = r'<br>'

print(re.split(rx, texto))
```

```
['uno', 'dos', 'tres']
```

re.sub

- **re.sub()** busca coincidencias y las reemplaza por unstring especificado.

```
import re

texto = r'precio unidad $20 total $200'

rx = r'\$'
```

```
print(re.sub(rx, "USD", texto))
```

```
precio unidad USD20 total USD200
```

Ejemplos en Python

Ejemplos

cambia \$ppp por USDddd

import re

def convierte(m):

peso = float(m.group()[1:])

return 'USD' + str(peso * 120)

texto = r'precio unidad \$20 total \$400'

rx = r'\\$\d+'

print(re.sub(rx, convierte, texto))

m es un match (Ej. en este caso será \$20 una vez, la segunda vez será \$400)

\$ seguido de uno o más dígitos
(funciona sólo para enteros)

precio unidad USD2400.0 total USD48000.0

Ejemplos

cambia \$ppp por USDddd - cotización en la función convierte es parámetro

```
import re
```

```
def convierte(m, c):
```

```
    peso = float(m.group()[1:])
```

```
    return 'USD' + str(peso * c)
```

```
texto = r'precio unidad $20 total $400'
```

```
rx = r'\$\d+'
```

```
cotiza = 120
```

```
print(re.sub(rx, lambda m: convierte(m, cotiza), texto))
```

Ejemplos

```
# elimina comentarios
```

```
import re
```

```
texto = r"""
```

```
# un programa Python
```

```
p = 5
```

```
print('El valor es: ', p * 2) # comentario
```

```
"""
```

```
rx = r'#.*'
```

← `.*` → uno o más caracteres que no sea fin de línea

```
print(re.sub(rx, "", texto))
```

Recursos

Ejemplos y explicación de expresiones útiles: <https://www.regular-expressions.info/examples.html>

Para probar expresiones regulares Python: <https://pythex.org/>

"Online RegEx tester and debugger": <https://regex101.com/>

Documentación oficial:

<https://docs.python.org/3/howto/regex.html#regex-howto>

<https://docs.python.org/3/library/re.html>