



Python Re Modülü

Python'da Düzenli İfade (Regular Expression) Kullanımı

Tarih: 14-01-2018



Düzenli ifadeler (Regular Expressions) yazılım hayatınızda mutlaka karşınıza çıkacak bir terimdir. Çok sık çıkmaz ama kesinlikle çıkar. Anlaması ilk önce biraz zor gelebilir, bazı temel yapıları uygulamalı olarak anlatmaya çalışalım. Zaten ben de bu içeriği, ihtiyacım olduğunda tekrar geri dönüp, konuyu hatırlayabilmek için yazıyorum.

Düzenli ifadeler tüm modern dillerde bulunur, dağınık bir metin içerisinde istedigimiz formattaklı metinleri yakalayabilmemize imkan tanır. Mesela, bir kaynacta geçen tüm e-posta adreslerini veya içinde rakam bulunan ve gmail uzantılı olan mail adreslerini ayıklamak için kullanabilirsiniz. Düzenli ifadeler olmasa ardi arkasına birçok if - else yazmak gerekebilirdi. Bu modül, birkag saatte yapabileceğiniz bir işlemi saniyeler içerisinde sizin yerinize yapabiliyor. Uzmanlaşması biraz vakit alsa da unutmasın en hızlı modüllerden biri bence.

Kurulum

Python ile standart olarak gelen bir kütüphanedir. Kurulum gerektirmez.

```
import re
```

şeklinde projeniz içeresine aktardıktan sonra kullanmaya başlayabilirsiniz.

search

Bu method, aranılan bir içeriğin ilgili metin içerisinde olup olmadığını kontrol eder.

```
>>> re.search("Sinan","Merhaba ben Sinan, senin adım nedir?")  
<_sre.SRE_Match object; span=(12, 17), match='Sinan'>
```

Metin içerisinde geçen Sinan kelimesini arattırdı, dikkat ederseniz span ve match diye alanlar var. Burada match aradığınız değeri, span ise nerede olduğunu gösterir. Aradığınız yerdeki 12. ve 17. harfler arasındaymış o zaman kontrol edelim.

```
>>> metin = "Merhaba ben Sinan, senin adım nedir?"  
>>> metin[12:17]  
'Sinan'
```

Gördüğünüz gibi tekrar Sinan değeri döndü. Search methodu ilgili metnin nerede olduğunu başarıyla bize söyleyebildi.

start()

Bu method aratılan kelimenin, kaynacta nerede geçtiğini döndürür. Biz zaten yukarıdaki örnekte (12,17) arasında olduğunu biliyoruz. Sadece burdan 12 değerini çekmek istersek kullanabiliriz.

```
>>> kontrol = re.search("Sinan","Merhaba ben Sinan, senin adım nedir?")  
>>> kontrol.start()  
12
```

end()

Üstteki start methodunun tersini yapar, yani aratılan kelime hangi aralıkta geçiyorsa onun son değerini döndürür. Biz kaynacta Sinan kelimesinin (12,17) arasında geçtiğini biliyoruz. Start methodu 12 dönmüşü, end methodu da 17 değerini dönecektir.

```
>>> kontrol = re.search("Sinan","Merhaba ben Sinan, senin adım nedir?")  
>>> kontrol.end()  
17
```

endpos()

Kaynak içerisindeki karakterlerin toplam sayısını döndürür. \n gibi satır atlatma karakterleri de sayılır.

```
>>> metin = "Merhaba ben Sinan, senin adım nedir?"  
>>> kontrol = re.search("Sinan",metin)  
>>> kontrol.endpos  
36  
>>> metin[:36]  
'Merhaba ben Sinan, senin adım nedir?'
```

İlk metin değişkeni içine bir içerik yazdım, sonra içinde sinan geçiyormu diye bakarak sonucu bir kontrol objesine atadım, ardından endpos ile 36 karakter olduğunu gördüm, sonra metin değişkenini 36'ya kadar görüntüledim ve doğruluğunu teyid ettim.

findall

Bu method ile bir kaynak içerisinde istedigimiz metnin kaç kere geçtiğini inceleyebiliriz.

```
>>> print(re.findall("zaman","Tam zamanında geldin, sensiz zaman geçmiyor"))  
['zaman', 'zaman']
```

Oluşturduğum metin içerisinde zaman kelimesi var mı diye sorguladım. Liste olarak geri döndü. Eğer bunu adet olarak görmek isteseydim len() methodundan geçirilebilirdim.

```
>>> print(len(re.findall("zaman","Tam zamanında geldin, sensiz zaman gecmiyor")))
2
```

Bu şekilde 2 kere geçtiğini görmüş oldum.

Metakarakterler

Bu noktaya kadar bir kaynak içerisinde belirli bir metin aradık, ancak bu modülü güçlü yapan şey bu noktadan sonra anlatacağımız metakarakterlerdir. Bu metakarakterler sayesinde, belirlediğiniz bir düzene uygun metinleri arayabilirsiniz. Mesela sinan metinini aramak yerine, sinan, sunan, suhan, solan gibi belirli bir şablon içerişine giren metinleri de arayabilirsiniz. İşte bunu yapmamızı sağlayan, aranılan metne eklediğimiz bazı özel karakterler var, bu karakterleri inceleyelim.

. Karakteri

Yeni satır karakteri hariç herhangi tek bir karakterin yerini tutar.

```
>>> re.findall(".aman","O her zaman, saman altından su yürütürdü.")
['zaman', 'saman']
```

Mesela bu örnek içerisinde zaman ve saman kelimelerini buldu. Çünkü herhangi bir karakter ile başlayıp aman ile devam eden bir şablon belirledik.

* Karakteri

Bir ifadenin bütün tekrarlamalarını bulur.

```
>>> re.findall("@g@mail","E-posta adresimiz test@mail.com, test@gmail.com ve test@g
['@mail', '@gmail', '@ggggggmail']
```

g harfinden sonra * koyduk, yani bu g karakteri 0 kere de geçebilir 100 kere de geçebilir. Bu ayrılm sadece kendinden önceki harfi kapsıyor. Bu nedenle @mail, @gmail ve @ggggggmail döndü.

+ Karakteri

Kendinden önce gelen karakterin bir veya daha fazla kullanılmasını arar. Üstte yaptığımız * ile ilgili örneği bu sefer + ile yaparsak;

```
>>> re.findall("@g+mail","E-posta adresimiz test@mail.com, test@gmail.com ve test@g
['@gmail', '@ggggggmail']
```

şeklinde @gmail ve @ggggggmail döner. Çünkü g harfinden sonra + koyarak bu harfin 1 veya daha fazla geçmesi gerektiğini söyledik.

? Karakteri

Kendinden önce gelen karakterin 0 veya 1 kere tekrar etmesini sorgular.

```
>>> re.findall("@g?mail","E-posta adresimiz test@mail.com, test@gmail.com ve test@g
['@mail', '@gmail']
```

Dikkat ettiyiniz @ggggggmail dönmemi çünkü 0 veya 1 kere olmasını sorguladık, 1'den fazla olanlar bu sonucu dahil değil.

[] Karakteri

Bu köşeli parantezler arasında yazılın bütün karakterler sorgulanır.

```
>>> re.findall("[HB]ilal","Hilal ve Bilal çok iyi arkadaştır.")
['Hilal', 'Bilal']
```

Bu şablonda H veya B ile başlayan sonrasında ilal ile devam eden bir şablon belirledik ve Hilal, Bilal sonuçlarını aldık.

```
>>> re.findall("s[a]i[m]e]it","Kardeşim samet, markete simit almaya gitti.")
['samet', 'simit']
```

Burda ise s ile başlayan a veya i artı m ile devam eden, e veya i artı t ile biten sonuçları aradık. Sonrasında samet ve simit sonuçlarını başarıyla gördük.

Biraz daha işleri karıştıralım. Köşeli parantez kullanırken aralık da belirleme imkanınız var.

- [A-Z] A'dan Z'ye tüm BÜYÜK harfler.
- [a-z] A'dan Z'ye tüm küçük harfler.
- [0-9] 0'dan 9'a tüm rakamlar.
- [1-4] 1'den 4'e tüm rakamlar.

O zaman madem Hilal ve Bilal büyük harf ile başlıyor, şöyle yazalım.

```
>>> re.findall("[A-Z]ilal","Hilal ve Bilal çok iyi arkadaştır.")
['Hilal', 'Bilal']
```

A-Z arasındaki büyük bir harfle başlayan ve sonra ilal ile devam edenleri bulduk. Peki küçük harfle arasıydı ne olurdu?

```
>>> re.findall("[a-z]ilal","Hilal ve Bilal çok iyi arkadaştır.")
[]
```

Gördüğünüz gibi hiçbir sonuç dönmedi. Çünkü Hilal ve Bilal büyük harfler ile başlıyor.

{ } Karakterleri

Birbirli bir sayıda tekrar anlamlıdır. Şimdi yukarıda öğrendiklerimiz ile birlikte biraz daha karışık bir örnek yapalım. Kaynağımız **Bence saat tamir etmek zor zanaat**. olsun. Burda küçük harflerle başlayan aat ile biten tüm herşeyi yakalamak istedik.

- Adım 1: [a-z] ile küçük harf ayırmam gereklidir.
- Adım 2: [a-z]* ile küçük harflerden 1 veya n (sonsuz) kere geçmesini söylemek.
- Adım 3: [a-z]+a ile küçük harf ile başlayıp a ile devam etsin dedik.
- Adım 4: [a-z]+a{2} ile a harfinden 2 adet olması gerektiğini belirtmek.
- Adım 5: [a-z]+a{2}t en son da t ile bitsin dedik.

```
>>> re.findall("[a-z]+a{2}t","Bence saat tamir etmek zor zanaat.")  
['saat', 'zanaat']
```

Bingo!

^ Karakteri

İfadenin başlangıcını kontrol eder ve [] karakterleri ile birlikte kullanılırsa da hariç anlamına gelir.

```
>>> re.findall("^Oku","Oku oğul, sesli oku.")  
['Oku']  
>>> re.findall("^sesli","Oku oğul, sesli oku.")  
[]
```

Kaynağımız Oku ile başladığı için ilk örnek başarıyla ~Oku sayesinde çalıştı. Ancak diğer örnekte ~sesli ayacı birsey dönmedi, halbuki kaynak içerisinde bu metin var fakat başında değil.

Bu meta karakter sabit bir metin için kullanışlı gelmeye bilir. Çünkü zaten metnin başına gözle de görülebiliyorsunuz, fakat bir listeyi for döngüsüne sokarak böyle bir metakarakter kullanılırsa çok faydalı olabilir. Bir örnek yapalım.

```
metin = "Mustafa başkomiser ve yardımcısı Kemalettin, 34XY6699 plakali arabanın peşini listeye = metin.split()  
for i in liste:  
    sonuc = re.findall("^[A-Z]+[a-z]+",i)  
    if sonuc:  
        print(sonuc)  
  
['Mustafa']  
['Kemalettin']
```

Örnek içerisinde metin değişkenindeki içeriği split() methodu ile kelime kelime parçalayıp liste içerisine attık. Sonra da bir for döngüsü ile her bir kelimeyi kontrol ederek büyük harfli kelimeleri ayırdık.

Şimdi de plakayı biraz uzun bir yol kullanarak bulalım.

```
for i in liste:  
    sonuc = re.findall("[^A-Za-z-][0-9]+[A-Z]+[0-9]+",i)  
    if sonuc:  
        print(sonuc)  
  
[ '34XY6699' ]
```

[] arasındaki A-Z-a ifadesi büyük veya küçük harf anlamını taşımıyor. Başına da ^ eklediğimde büyük veya küçük harf ile **başlamasın** demiş olduk. Sonra 0-9 ile devam etsin sonra tekrar büyük harf ve sonra tekrar 0-9 ile devam etsin. Sonuç olarak plakayı çıktıktı.

Peki plakayı daha kısa bir yoldan bulmak isteseydik nasıl bulurduk? Bu da ödev olsun yorum olarak yazarsınız.

§ Karakteri

Yukarıda, ^ karakteri ile bir ifadenin başlangıcını kontrol etmiştık, \$ metakarakteri ile de ifadenin sonunu yani ne ile bittiğini kontrol edebiliyoruz.

```
metin = "Silikon vadisi, google.com ve apple.com arasındaki rekabeti tartışıyor."  
liste = metin.split()  
for i in liste:  
    sonuc = re.search(".com$",i)  
    if sonuc:  
        print(sonuc.string)  
  
google.com  
apple.com
```

Örnek bir metin oluşturup içerisinde google.com ve apple.com alan adlarını yerleştirdim. Ardından bu metni split() methodu ile bir liste içerisine aldım. Sonra bir for döngüsü ile her bir kelimenin .com ile bitip bitmediğini, eğer bitiyorsa ekrana yazmasını istedim. Bu şekilde bir metinde geçen .com ile biten kelimeleri çıkarabildim.

Daha anlatmadım, aşağıda anlatacağım ama yeri gelmişken yazmayı istedim. Eğer burdaki .com veya .net ile bitenleri bulmak istersek aradaki **veya** ifadesini karşılamak için | karakterini kullanıyoruz. Örnek:

```
metin = "Silikon vadisi, google.com ve apple.net arasındaki rekabeti tartışıyor."  
liste = metin.split()  
for i in liste:  
    sonuc = re.search("(\\.com|\\.net)$",i)  
    if sonuc:  
        print(sonuc.string)
```

I Karakteri

Yukarıda verdiğim örnek içerisinde kullanılmışım, veya anlamına gelir.

```
>>> re.findall("(siyah|beyaz)", "Beşiktaş, siyah ve beyaz renkleri ile anılır.")  
['siyah', 'beyaz']
```

Kaynaktan siyah veya beyaz metinlerini aldı. Arama yaparken birden fazla değeri kaynak içerisinde bulmayı mümkün kılar.

() Karakterleri

Parantez de yazdığımız kalıpları grüplamak için kullanılır. Matematikteki bölme ve çarpmaya için öncelik belirlerken kullandığımız parantez gibi düşünebilirsiniz.

\ Karakteri

Kaçış dizisidir. Buraya kadar birçok karakter gördük, mesela bunlardan biri de \\$ simgesi. Bir ifadenin sonunu kontrol etmek için kullanıyoruz. Ancak bu bir dolar simgesi ve bazen bir para birimi olarak da kaynak içerisinde bulunabilir, ve biz bunları ayıklamak isteriz. Mesela;

```
>>> re.findall("[0-9]+$","Ekran kartı 150$, ses kartı ise 90$")  
[]  
>>> re.findall("[0-9]+\$","Ekran kartı 150$, ses kartı ise 90$")  
['150$', '90$']
```

0-9 ile başlayan sonra \$ ile devam eden bir arama yaptığımızda eğer ters slash koymazsa python bunun özel bir anlam ifade ettiğini zannediyor ve sonuç döndürmüyorum. Kaçış karakteri kullandığımızda ise başarıyla fiyatları döndürüyor.

Meta Sequences

Bunu nasıl çevireceğimi bilemedim. Metakarakter gibi ama özel bir anlam ifade eden yapılar var son olarak onlara bakalım.

\s ve \S İfadesi

Bu ifade kaynaktaki boşluğu yakalar. Büyük harfle yazılan hali ise boşluk haricindekileri yakalar.

```
>>> re.findall("\s","Bil bakalım kaç boşluk var.")  
[' ', ' ', ' ', ' ']  
>>> len(re.findall("\s","Bil bakalım kaç boşluk var."))  
4
```

Metin içinde kaç tane boşluk olduğunu görmek için \s ifadesini kullandım. Sonra len methodu ile adedini ekrana bastırdım.

Peki boşluk haricindekileri bulmak isteseydim? Onu da \S ile yapıyorum.

```
>>> re.findall("\S+","Bil bakalım kaç boşluk var.")  
['Bil', 'bakalım', 'kaç', 'boşluk', 'var.']
```

\d ve \D İfadesi

Bu ifade kaynaktaki sayıyı yakalar. Büyük harfle yazılan hali ise sayı olmayanları yakalar.

```
>>> re.findall("\d+","Gelirken 1 kilo yoğurt ve 2 adet ekmek alırsın.")  
['1', '2']
```

Başarıyla kaynaktaki sayıları yakaladım. Sayı haricindekileri yakalayalım.

```
>>> re.findall("\D+","Gelirken 1 kilo yoğurt ve 2 adet ekmek alırsın.")  
['Gelirken', ' kilo yoğurt ve ', ' adet ekmek alırsın.']
```

\w ve \W İfadesi

Bu ifade kaynaktaki karakter,sayı ve alt çizgiyi yakalar. Büyük harfle yazılan hali ise tam tersini yani boşluk, nokta, soru işaretleri, boşluk gibi karakterleri yakalar.

```
>>> re.findall("\w+","Mail adresin sinan%erdinc@test.com mu?")  
['Mail', 'adresin', 'sinan', 'erdinc', 'test', 'com', 'mu?']  
>>> re.findall("\W+","Mail adresin sinan%erdinc@test.com mu?")  
[' ', ' ', '%', '@', '.', ' ', '?']
```

\w kullandığında % @ ? simgeleri gelmezken \W kullandığında geliyor.

Not: Ek olarak hızlıca regex denemeleri yapmak isterseniz <https://regex101.com/> ve <https://rerexr.com/> adreslerini kullanabilirsiniz.



← ÖNCESİ YAZI

SONRAKİ YAZI →

[sinanerdinc.com Yorum İknesi](#)

Yorum yazmak, bu konu hakkında özgürce fikirlerinizi paylaşmanıza olanak tanır.



[Oner 3](#) [Tweet Gonder](#) [Paylas](#)

En Iyiye Gore Sırala ▾

Tartışmaya katıl...

SÜNUNLA OTURUM AÇIN YA DA BİR DISQUS HESABI AÇIN [\(?\)](#)

D F T G Ad

 **Berk** - 10 gün önce
Teşekkürler.
^ | v - Yanıtla - Paylaş >

 **oğulcan** - 17 gün önce
Merhaba kısa yoldan böyle yaptım programlamada yeniyim başka bir yöntemi varsa söyleyinim.
sevinirim.

```
import re
metin = "Mustafa başkomiser ve yardımcısı Kemalettin, 34XY6699 plakali arabanın peşinde."
liste = metin.split()
for i in liste:
    sonuc = re.findall("[^0-9]+[A-Z]+[0-9]+", i)
if sonuc:
    print(sonuc)
```

 **Ülkeyd Alkan (Salube)** - 8 ay önce
Çok güzel anlatmışsınız.
^ | v - Yanıtla - Paylaş >

[Abone Olun](#) [Sitenize Disqus ekleyin](#) [Verilerim Satılmışın](#)

DISQUS



Sinan Erdinç • 2020 • sinanerdinc.com
[sitemap](#)