Regular Expressions(Düzenli İfadeler)

Normal ifade veya "RegEx", verilerdeki kalıpları bulmaya yardımcı olan özel bir metin dizesidir. Bir "RegEx", farklı bir veri türünde bazı desenlerin olup olmadığını kontrol etmek için kullanılabilir. Python'da "RegEx"'i kullanmak için önce re adlı "RegEx" modülünü içe aktarmalıyız.

The re Module

Modülü içe aktardıktan sonra, kalıpları tespit etmek veya bulmak için kullanabiliriz.

import re

## Methods in re Module

Farklı re karakterleri kullanırız bazıları aşağıda verilmiştir.

* re.match() : yalnızca dizenin ilk satırının başında arar ve bulunursa eşleşen nesneleri döndürür, aksi takdirde None değerini döndürür.
* re.search : Çok satırlı dizeler de dahil olmak üzere dizenin herhangi bir yerinde varsa bir eşleşme nesnesi döndürür.
* re.findall : Tüm eşleşmeleri içeren bir liste döndürür
* re.split : Bir dize alır, onu eşleşme noktalarında böler, bir liste döndürür
* re.sub : Bir dizgedeki bir veya daha fazla eşleşmeyi değiştirir

## Match

import re

txt='I love to teach python and c++.'

match= re.match('I love to teach',txt,re.I)

print(match)

#<re.Match object; span=(0, 15), match='I love to teach'>

span=match.span()

print(span)#0, 15)

start,end=span

substring = txt[start:end]

print(substring)

# I love to teach

Yukarıdaki örnekten de görebileceğiniz gibi, aradığımız kalıp (veya aradığımız alt dizi)

“I love to teach” eşleştirme işlevi, **yalnızca**metin desenle başlıyorsa bir nesne döndürür.

txt = 'I love to teach python and javaScript'

match = re.match('I like to teach', txt, re.I)

print(match)

#None

Dize “I love to teach “ ile dize değil , bu nedenle eşleşme olmadı ve eşleşme yöntemi None döndürdü.

## Search

import re

txt = '''Python is the most creative programming language. Over many programming languages, I recommend Python.'''

match = re.search('many', txt, re.I)

print(match)

#<re.Match object; span=(55, 59), match='many'>

span = match.span()

print(span)

(55, 59)

start, end = span

print(start, end)

 #55 59

substring = txt[start:end]

print(substring)

 #many

Gördüğünüz gibi arama, eşleşmeden çok daha iyidir çünkü metin boyunca deseni arayabilir. Arama, bulunan ilk eşleşmeyle bir eşleşme nesnesi döndürür, aksi takdirde None döndürür. Çok daha iyi bir “re” işlev findall. Bu işlev, tüm dize boyunca kalıbı kontrol eder ve tüm eşleşmeleri bir liste olarak döndürür.

## Searching for All Matches Using findall(Findall Kullanarak Tüm Eşleşmeleri Arama)

findall() tüm eşleşmeleri bir liste olarak döndürür

txt = '''Python is the most creative programming language. Over many programming languages, I recommend Python.'''

matches = re.findall('language', txt, re.I)

print(matches)

#['language', 'language']

Gördüğünüz gibi "language" kelimesi dizede iki kez bulundu. Biraz daha pratik yapalım. Şimdi dizede hem “Python” hem de “python” kelimelerini arayacağız:

txt = '''Python is the most creative programming language. Over many programming languages, I recommend Python.'''

matches = re.findall('python', txt, re.I)

print(matches)

['Python', 'Python']

“python” eşleştirme sonucu “Python” larda eşlemeye dahil oldular “re.I” kullanımı büyük ve küçük harfleri aramaya dahil eder.

## Replacing a Substring(Bir alt dizinin değiştirilmesi)

Bir örnek daha ekleyelim. % sembolünü kaldırmadıkça aşağıdaki dizeyi okumak gerçekten zor. % öğesini boş bir dizeyle değiştirmek metni temizleyecektir.

txt = '''h%e%l%l%o, %I %d%o %d%a%t%a %a%n%a%l%y%s%i%s. %I %u%s%u%a%l%l%y %u%s%e %p%y%t%h%o%n%.'''

matches = re.sub('%', '', txt)

print(matches)

## Splitting Text Using RegEx Split(RegEx Split Kullanarak Metni Bölme

txt=''' hello, I do data analysis. I usually use python.

Python is the most creative programming language.

Over many programming languages, I recommend Python.'''

print(re.split('\n', txt))

#[' hello, I do data analysis. I usually use python.',

# 'Python is the most creative programming language. ',

#  'Over many programming languages, I recommend Python.']

## Writing RegEx Patterns(RegEx Kalıpları Yazma)

Bir dize değişkeni bildirmek için tek veya çift tırnak kullanırız. RegEx değişkeni r'' bildirmek için . Büyük/küçük harfe duyarsız hale getirmek için kalıbımızı yeniden yazmalıyız veya bir bayrak eklemeliyiz.

import re

regex\_pattern = r'banana'

txt = 'I love melon and banana.There is a lot of vitamin E in apples. There is vitamin K in banana. '

matches = re.findall(regex\_pattern, txt,re.I)

print(matches)

## ['banana', 'banana']

## Regular Expression Character

import re

regex\_pattern = r'banana'

txt = 'I love melon and banana.There is a lot of vitamin E in

apples. There is vitamin K in banana. '

matches = re.findall(regex\_pattern, txt,re.I)

print(matches)

##['banana', 'banana']

* []: Bir dizi karakter

[ac] a veya b veya c anlamına gelir

[az] a'dan z'ye herhangi bir harf anlamına gelir

[AZ], A'dan Z'ye herhangi bir karakter anlamına gelir

[0-3], 0 veya 1 veya 2 veya 3 anlamına gelir

[0-9] 0'dan 9'a kadar herhangi bir sayı anlamına gelir

[A-Za-z0-9] herhangi bir tek karakter, yani a'dan z'ye, A'dan Z'ye veya 0'dan 9'a

* : özel karakterlerden kaçmak için kullanır

\d şu anlama gelir: dizenin rakamları içerdiği yerde eşleşme (0-9 arası sayılar)

\D şu anlama gelir: dizenin rakam içermediği yerde eşleşme

. : yeni satır karakteri (\n) dışında herhangi bir karakter

^: ile başlar

* r'^substring' örneğin r'^love', aşk kelimesiyle başlayan bir cümle
* r'[^abc], a değil, b değil, c değil anlamına gelir.
* $: ile biter
* r'substring$' örneğin r'love$', aşk kelimesiyle biten cümle
* \*: sıfır veya daha fazla kez

r'[a]\*' isteğe bağlı anlamına gelir veya birçok kez ortaya çıkabilir.

+: bir veya daha fazla kez

* r'[a]+' en az bir (veya daha fazla) anlamına gelir

?: sıfır veya bir kez

r'[a]?' sıfır kez veya bir kez anlamına gelir

* {3}: Tam olarak 3 karakter
* {3,}: En az 3 karakter
* {3,8}: 3 ila 8 karakter
* |: Ya veya

\*r'apple|muz' ya elma ya da muz anlamına gelir

* (): Yakala ve grupla

## Square Bracket(Köşeli Ayraç)

import re

regex\_pattern = r'[Bb]anana'

txt = 'Banana is a storehouse of vitamins .There is a lot of vitamin E in apples. There is vitamin K in banana. '

matches = re.findall(regex\_pattern, txt)

print(matches)

#['Banana', 'banana']

## RegEx'te kaçış karakteri ()

regex\_pattern = r'\d'

txt = 'TGNA OPENED ON APRIL 23, 1920'

matches = re.findall(regex\_pattern, txt)

print(matches)

# ['2', '3', '1', '9', '2', '0']

## One or more times(+)(Bir veya daha fazla kez)

egex\_pattern = r'\d+'

txt = 'TGNA OPENED ON APRIL 23, 1920'

matches = re.findall(regex\_pattern, txt)

print(matches)

#['23', '1920']

## Period(.)

regex\_pattern = r'[a].'

txt = '''Apple and banana are fruits'''

matches = re.findall(regex\_pattern, txt)

print(matches)

#['an', 'an', 'an', 'a ', 'ar']

regex\_pattern = r'[a].+'

matches = re.findall(regex\_pattern, txt)

print(matches)

#['and banana are fruits']

## Zero or more times(\*)

regex\_pattern = r'[a].\*'

txt = '''Apple and banana are fruits'''

matches = re.findall(regex\_pattern, txt)

print(matches)

#['and banana are fruits']

## Zero or one time(?)

## Sıfır veya bir kez. Model oluşmayabilir veya bir kez oluşabilir.

txt = '''hello, I do data analysis. I usually use python.

Python is the most creative programming language.

Over many programming languages, I recommend Python.'''

regex\_pattern = r'[Pp]-?ython'  # ? means here that '-' is optional

matches = re.findall(regex\_pattern, txt)

print(matches)

# ['python', 'Python', 'Python']

## Quantifier in RegEx

Bir metinde aradığımız alt dizenin uzunluğunu küme parantezini kullanarak belirtebiliriz. 4 karakter uzunluğunda bir alt diziyle ilgilendiğimizi düşünelim:

txt = 'Python development began in 1990. Python reached version 1.0 in January 1994.'

regex\_pattern = r'\d{4}'

matches = re.findall(regex\_pattern, txt)

print(matches)

#['1990', '1994']

## Cart ^

* Starts with

Cümlenin başladığı kelimeyi inceler.

* Negation

Olumsuzlama anlamına gelir.

txt = 'Python development began in 1990. Python reached version 1.0 in January 1994.'

regex\_pattern = r'[^A-Za-z ]+'

matches = re.findall(regex\_pattern, txt)

print(matches)

#['1990.', '1.0', '1994.']