Python'da Düzenli İfadeler – II

Python'da düzenli ifadeler ile ilgili konuların büyük bir çoğunluğunu önceki notta anlatmıştım. Bu notta, bir önceki notta eksik kalan bazı noktaları ve bazı ipuçlarını paylaşacağım.

" ".join( ) methodu:

Bu method, join'in içine aldığı listin elemanlarını bu elemaşnarın arasına boşluk koyup birleştirerek yeni bir string oluşturur.

```
>> listem =["Mumkun", "oldugunca", "evde", "kalin"]

>> " ".join(listem)
'Mumkun oldugunca evde kalin'
```

Burada elemanlarını birleştirmek istediginiz listenin elemanlarini boşluk ile birleştirmek zorunda değilsiniz. Hangi karakter yada karakterlerle birleştirmek isterseniz onu `" ".join()` 'da tirnağin icine yazmaniz yeterli. Örnegin diyelimki yukaridaki cumleyi _ ile birleşirmek isteyelim.

```
>> "_".join(listem)
'Mumkun_oldugunca_evde_kalin'
```

re.sub() fonksiyonu:

`re.sub()` fonksiyonu metin içinde belirli bir kelime yada belirli bir yapı (pattern) ile eşleşme durumunda bu kelimenin yada yapının istedğimiz karakterle yer değiştirmesi için kullanılır. Fonksiyonun adındaki `sub`, "substitute" yani yerine geçmek fiilinden gelmektedir. Bu fonksiyonun genel formu şöyledir:

```
re.sub(<aranacak yapı>,<yerine geçecek karakter(ler)>," arama yapılan metin ")
```

```
>> re.sub("a{4}","a","Ziyaaaa!") # üstüste dort a bul, onun yerine bir a yaz.
Ziya!
```

Şimdi de bir metindeki tüm boşlukları _ ile dolduralım.

```
>> met = "bir bosluk  iki bosluk   uc bosluk"

>> re.sub(" +","_",met) #bir ve daha  fazla boşluk varsa _ ile doldur
'bir_bosluk_iki_bosluk_uc_bosluk'

>> re.sub("\s+","_",met) #bu da alternatif bir cozum

'bir_bosluk_iki_bosluk_uc_bosluk'
```

re.split() fonksiyonu:

`re.split()` fonksiyonu metini belirli bir kelimeye göre yada belirli bir yapıya (pattern) göre ayrıştırmaya ve bu ayrışan parçaları bir liste halinde sunmaya (döndürmeye) yarar. Genel formu şöyledir:

```
re.split(<ayrıştırmanın neye göre olacağı>," arama yapılan metin ")
```

```
>> cumle = "Bugun#benim#dogum#gunum"

>> cumle = re.split("#",cumle)
['Bugun', 'benim', 'dogum', 'gunum']

>> cumle = "BugunAbenimAAdogumAAAgunum"
```

```
>> cumle = re.split("A+",cumle)
['Bugun', 'benim', 'dogum', 'gunum']
```

```
>>re.split("[aeiuo]", "Sivas")  #Sivas kelimesini sesli harflere göre ayıralım
['S', 'v', 's']
```

\w ifadesi

`\w` ifadesi metindeki her harf, her sayı ve her alt çizgi ile eşleşir, noktalama işaretlei ve boşlukla eşleşmez.

```
>>re.findall("\w","Yas 35! Yolun yarisi eder.")
['Y','a','s','3','5','Y','o','l','u','n','y','a','r','i','s','i','e','d','e','r']
```

`\w` ifadesi tek bir karakterle eşleşir. Bosluk, yada bir noktalama isareti gelene kadar birden çok karakterle eşlessin istersek `\w+` ifadesini kullanabiliriz.

```
>>re.findall("\w+","Yas 35! Yolun yarisi eder.")
['Yas', '35', 'Yolun', 'yarisi', 'eder']
```

`\W` ifadesi tam zıt olarak bosluk ve noktalama işareti ile eşleşir.

```
>>re.findall("\W","Yas 35! Yolun yarisi eder.")
[' ', '!', ' ', ' ', ' ', '.']
```

Notlar:

1. `.+ (veya .*)` her şeyle eşleşir:

```
>>re.findall(".+","Yas 35! Yolun yarisi eder.")
['Yas 35! Yolun yarisi eder.']
```

2. `\w*` boşluk ve noktalama işareti hariç herşeyle eşleşir. Örneğin bir önceki örnekte içinde a harfi olan kelimleri bulalım:

```
>>re.findall("\w*a\w*","Yas 35! Yolun yarisi eder.")
['Yas', 'yarisi']
```

3. `[a-zA-Z0-9]` tüm küçük harfler, yada tüm büyük harfler yada tüm sayılar demektır.

```
>>re.findall("[a-zA-Z0-9]*a[a-zA-Z0-9]*","Yas 35! Yolun yarisi eder.")
['Yas', 'yarisi']
```

4. `{}` karakterini, bir önceki notta kendinden önce gelen karakterin kaç defa gelebileceğini göstermek için kullanmıştık. Örneğin `a{4}, dört` tane yanyana gelena ile eşleşir (bkz. yukarıdaki Ziyaaaa örneği) . Özel olarak eğer `{n,m}` şeklinde kullanırsak bu ifade kendinden önce gelen karakter en az n, en çok m defa görüldüğünde eşleşir.

Simdi diyelimki bir metinde en az 4 harfli, en çok 5 harfli kelimeleri bulalım:

```
>>re.findall("[A-Za-z]{4,5}","ucc dort besss altiii")
['dort', 'besss', 'altii']
```

Bu şekilde altiii kelimesinin ilk 5 harfini de aldı; bunu istemiyorsak `[A-Za-z]{4,5}` düzenli ifadesinin sonuna bir boşluk koyarız. Bu dört yada 5 tane harften sonra bir boşluk gelmek zorundadır anlamı taşır:

```
>>re.findall("[A-Za-z]{4,5} ","ucc dort besss altiii")
['dort ', 'besss ']
```

Eğer özel olarak `{n,}` yazarsak bu, karakterin en az n kere tekrarlanacağı anlamına gelir; `{,m}` yazarsak bu, karakterin en fazla `m` kere tekrarlanacağı anlamına gelir.

```
>>re.findall("[A-Za-z]{4,}","ucc dort besss altiii")  #en az 4 harfli kelimeler
['dort', 'besss', 'altiii']
```

```
>>re.findall("[A-Za-z]{,5} ","ucc dort besss altiii")  #en az 4 harfli kelimeler
['ucc ', 'dort ', 'besss ']
```

5. `\d` ifadesinin metindeki rakamlarla (digit'lerle) eşleştiğini öncekini notta görmüştük. `\d` ifadesi bu haliyle rakamlarla tek tek eşleşir; bütün bir sayı ile eşleşmesini istiyorsak , en az bir ve daha fazla rakam anlamına gelen `\d+` ifadesini kullanmamız gerekir.

```
>> re.findall("\d+"," Universitemizin telefon numarasi: 0346 219 10 10")
['0346', '219', '10', '10']
```

```
>> re.findall("[0-9]+"," Universitemizin telefon numarasi: 0346 219 10 10")

['0346', '219', '10', '10']
```

6. `\s` ifadesinin metindeki boşluklarla eşleştiğini görmüştük. Ama `\s` bu haliyle her bir boşluk karakteriyle tek tek eşleşir. Eğer çeşitli uzinluklardaki boşluklar ile eşleme yapmak istiyorsak `\s+` ifadesini kullanırız.

```
>> re.findall('\s+',"bir iki  uc   dort    ")
[' ', '  ', '   ', '    ']
```

```
>> re.split('\s+',"bir iki  uc   dort")
['bir', 'iki', 'uc', 'dort']
```

Yukarıdaki son örnekte kelimeler arası boşluk ne kadar olursa olsun, metini kelimelerine ayırabildik.