

PHP - Regular Expression

Regular Expression nedir ?

Yazımın bundan sonraki bölümlerinde Regex diye anacağım (Regexp diye kısaltılır ama okunuşu kolaylaştığı için Regex diyeceğim.) Regular Expression (Düzenli İfadeler) , belli kuralı olan karakter topluluklarının bir görüntüsünü (desen) yazarak tanımlayabileceğimiz ya da kontrolünü yapabileceğimiz bir dil olarak kabul edilebilir. Belli kuralı olan karakter topluluğu diye bahsettik ama eğer bir kuralı zaten varsa bunu programlamaya neden ihtiyaç duyuyoruz? Diyebilirsiniz. Esnek bir kuralı varsa bu toplulukların, işte o zaman Regex bizim için eşi bulunmaz bir yardımcı olacaktır.

Elimizde bir figür olduğunu düşünelim. Bu figürün bir insan şekline denk gelip gelemeyeceğini nasıl tespit edebilirsiniz? Biyolojik zorunluluklar dışında bir insan şeklini tanımlamanın şartları şunlar olabilir mi ? Regex, soyut kavramları tanımlamada kullanılmadığı için vereceğim örnekler görüntü ile alakalı olacaktır.

- 2 ayak, 2 el (her elde 5 parmak), gövde, baş, 2 göz, 2 kulak

Normal bir insan bu tanıma uyar mı? Evet, uyar ve insan ayırt etmeden bu görüntü her insan için geçerlidir. Şimdi insanlar arasından seçeceklerimize görünümle ilgili birkaç kıstas daha ekleyelim

- Minimum boy: 180 cm

İnsanların çoğunu bu kıstaslarla eleyebiliriz aradığımız insanı bulmaya doğru gidiyoruz.

- Gözlük, Şapka(olmayabilir)

2 ölçüt daha ekledik fakat ölçütlerin biri seçmeli oldu yani eleme işini o yapamayacak. Gözlük şartıyla yine elemiş olduk

İnsanlar arasından aradığımız insan modeline uyanları seçmek için kriterler belirledik ve aradığımıza gidebilmek için kriterleri arttırdık hatta tam sayılar vererek katılaştırdık. Regex'te de kurallarınız ne kadar katıysa o kadar kesin, bir o kadar da az sonuç alırsınız.

Regex Deseni oluşturmadan önce

Regex desenleri oluşturmaya başlamadan bir önceki metinde değindiğimiz kriterleri tek tek yazmanız ve bulabileceğiniz sonuçları tartmanızda fayda vardır. İnsan modelleme örneğine başlamadan önce şu soruları sorabilirdik: 2 ayağı olan kaç insan bulurum? , 2 gözü olan kaç insan bulurum ?. bunlar insanda ayır edici özellik değildir. Bulmak ya da elemek istediğimiz elemanlarda ayırt edici özellik aramak bizim için en kolay yöntem olacaktır. Bunları desen oluşturmadan önce tartışmamız, bazen aramadıklarımızın aradıklarımızdan daha az olduğunu gösterebilir. İsminde ünlülerden sadece "aeioöu" harflerden birini içerenleri taramak mı kolay sizce "ü" geçmeyenleri taramak mı? Regexte bizi hızlandıracak nokta bu olacaktır. Aradıklarımız mı? Aramadıklarımız mı?

Meta Karakterler

^	Metin Başlangıcı (multiline modunda satır başı)
\$	Metin Sonu (multiline modunda satır sonu)
[Karakter sınıfı tanımlama başlangıcı
]	Karakter sınıfı tanımlama sonu
 	Alternatif ayırıcı, örneğin (a b) hem a ile hem de b ile eşleşebilir
(Desen grubu başlangıcı
)	Desen grubu sonu
\	Kaçış karakteri (üstte tanımlanan ([\$ gibi karakterlerin desenimizde özel anlamları dışında sadece karakter olarak kullanılmasında yardımcı olur)

Ön Tanımlı Karakter Sınıfları

\w	Alfanumerik bir karakteri temsil eder [a-z0-9_]
\W	Alfanumerik karakterler dışında kalan bir karakteri temsil eder [^a-z0-9_]
\s	1 birim boşluğu temsil eder (boşluk, tab,satır sonu)
\d	Numerik bir karakteri temsil eder [0-9]
\D	Numerik olmayan bir karakteri temsil eder [^0-9]
.	Herhangi bir karakterin yerine geçebilir

Niceleyiciler

x*	Sıfır ya da daha fazla x
x+	Bir ya da daha fazla x
x?	Sıfır ya da bir tane x
x{n}	N tane x
x{n,}	En az n tane x
x{n,m}	En fazla m tane x

Desen Niteleyicileri (Pattern Modifiers)

i	Büyük küçük harf duyarlılığını devre dışı bırakır (Ignore case sensitive)
m	Multiline modu ^ ve \$ satır başı ve satır sonunu temsil eder (bkz: meta karakterler ^\$)
s	Dotall modu - nokta "." Set içinde kullanıldığında satır sonunu da temsil eder
x	Yorum ve boş satır sonlarını görmezden gelir
e	Eval modu .Sadece replace işlemlerinde geçerlidir.

Sınır İşaretçileri

\b	Kelime sınırı
\B	Kelimeler dışında kalan eşleşmeler sınırı
\A	Eşleşme başlangıcı
\Z	Eşleşme sonu ya da satır sonu
\z	Eşleşme sonu
\G	Eşleşmedeki başlangıcın indisi

Alt Desen Niteleyicileri ve İşaretçiler

(?:)	Alt eşleşme olmama şartı Örn : ((?:tufan baris)yildirim) diğer alt desenlerle eşleşen bir tufan ya da baris bulamadığı taktirde tufanyildirim ya da barisyildirim ile eşleşebilir.
(?=)	Kendisinden sonraki eşleşme geçerliye öncesini de eşleştirir tufan(=baris) hemen ardından baris içeriyorsa tufan ile eşleşir.
(?!)	Önceki işaretçinin tersidir kendisinden sonraki eşleşmiyorsa bu eşleşme kabul edilir
(?<=)	diğer işaretçilerle aynı işlemleri kendisinden öncekiler için yapar
(?<!)	Önceki işaretçinin tersidir
(?x)	Bu gruptaki ilk işaretçi için alt desen numarası ile işlem yapabilir Örn : (?2)tufan baris)yildirim bu gruptaki 2inci alt grup tufan ya da baris ise burası yildirim ile eşleşebilir.
(?#)	Yorum grubudur (?# bu kısım yorumlanmayacaktır açıklama yazabilir.)
--	(?isim) , (?'isim') (?Pisim) Bu işaretçiler de grupları adlandırmaya yarar php'nin 5.2.2 sürümünden öncesinde sadece <?P<isim>> çalışır. Regexp standartlarında ilk örnek olan (?isim) geçerlidir.

PHP'de Regular Expressions

PHP , açık kaynak bir kütüphane olan PCRE (Perl – Compatible Regular Expressions) kütüphanesini kullanır .PHP dilinin esnek yapısı bize Regexte de kolaylık ve esneklik sağlıyor. Regular expression replace terimlerinde eval (e) modunun olması onu diğer bütün dillerden farklı kılmıştır. Perl paketindeki tüm özelliklerle beraber bize bu kolaylığı sağlamasıyla beraber bizim yerimize Regex işlemlerini de önışlemden geçirip kullanımı oldukça esnekleştirmiş. Örnekler bunu daha net gösterecektir.

PHP'de Regex kullanmamızın bize sağlayacağı faydaların bir kaçını yazmak istersek.

- Metin Arama.
- Form Doğrulama.
- Form verileri güvenliği
- Metin Formatlama

PHP Regular Expression Fonksiyonlar (preg_ paketi)

preg_match	verdiğimiz pattern(desene)' e uyması durumunda eşleşme sayısı döner. Opsiyonel olarak verilen \$matches dizi parametresine ilk eşleşmeyi (alt eşleşmelerle) atabilir.
preg_match_all	verdiğimiz metin içerisinde desene uymayan eleman bulana kadar tekrar çalışır. Eşleşme bulunması durumunda eşleşme sayısını dönderir ve yine preg_match te olduğu gibi \$matches dizisine eşleşmeleri doldurabilir fakat bu fonksiyonda bu parametre zorunludur.
preg_replace	desene uyan bölümleri vereceğimiz değerlerle değiştirir.
preg_replace_callback	preg_replace ile aynı işi yaparken diziye doldurmadan önce her elemanı verdiğimiz fonksiyona parametre olarak vererek dönen değeri diziye alır.
preg_filter	preg_replace ile aynı işlemi yaparken sadece eşleşen kısımları dönderir.
preg_grep	verilen diziden , sadece verdiğimiz desene uyanları geri dizi olarak verir.
preg_split	explode() fonksiyonunun regex eşleşmesiyle çalışan şeklidir diyebiliriz. Desene uyan her bölümü silere kalan parçaları dizi olarak dönderir. Regex kullanmadan calama yapabileceğimiz diğer fonksiyonlar da explode() ve str_split() tir.
preg_quote	regex için açıkladığımız özel karakterler (metalar ve işaretçiler) tektek escape edilmek yerine preg_quote ile hatasız ve kolay bir şekilde escape edilebilir. Bunu kullanmamız hem kod içindeki okunurluğu arttırır hem de kodun başka dillere taşınmasında kolaylık sağlar Regex özel karakterleri : “. \ + * ? [^] \$ () { } = ! < > : -"
preg_last_error	bize desenimizde ya da desenin uygulamasındaki son hatayı verir.

Meta Karakterler

^ ve \$ (Satır başı ve satır sonu)

```
$metin="Regular Expression
satır sonu ve
Satır başını
deniyoruz ";
$eslesti=preg_match('/^.+$/m',$metin,$seslesme);
if ($eslesti) // sıfırdan büyük olmadı durumunda true sayılacaktır.
print_r($seslesme);
```

bu iki karakteri tek başına kullanmak örneğin anlaşılabilirliğini öldürebilirdi. Bu yüzden ilk örneğimizde meta karakterlerden bolca kullandık “^” , “.” , “+” ve “\$” modifier olarka /m verdik.

^ karakteri metnin başladığını \$ karakteri de bittiğini gösterir değil ama burada metnimizde bu kurallara uyan bir şey yok (/m eklemesek). Kuralımız şunu anlatıyor. Metnin başlangıcından sonra en az 1 karakter olacak “.” bunun devamında da metin bitecek. Kuralımızı biraz genişlettik. Multiline modifiyeri ile birlikte metnimizin çok satırdan oluştuğunu ve \$ karakterinin satır sonunu belirttiğini söyledek. Şimdi eşleşme alabiliyoruz ve \$seslesme dizisindeki elemanımız :

0 =>Regular Expression

Neren 1 eleman ? Regexp_match tanımlamamızda ilk eşleşmenin döneceğini belirtmiştik fakat subpattern kullanımlarında 1 eleman eşleşmenin tamamı diğer elemanlar da subpatternlar olur. Ama bu yine 1 eşleşmedir. Aynı örneği preg_match_all ile denersek

Array (

```
[0] => Array (
  [0] => Regular Expression
  [1] => satır sonu ve
  [2] => Satır başını
  [3] => deniyoruz ) )
```

Artık her eşleşme için bir dizi ve her eşleşmenin içindeki eşleşmeler de 1 dizi olarak tutuluyor. Burada ne kadar satır yazarsak yazalım bu kuralı bozamayacağımız için (. her hangi bir karakterdir) her zaman 1 tane eşleşme alırız içinde de satırlarımız olur.birden fazla eşleşmelerin olduğu örnekleri de incelemeye çalışacağız.

Karakter Sınıfları []

Regex bize arayacağımız elemanlar arasında alternatif tanımamızı “ | ” karakteri ile sağlamış fakat bunun daha pratik yolu olan ön tanımlı ve kendi tanımlamalarımızı yapabileceğimiz karakter sınıfları var. Örnek verecek olursak A dan Z ye 26 karakterimiz var (ön tanımlı bu sınıf Türkçe karakterleri içermiyor) bu 26 karakteri “|”(pipe) karakteri ile ayırmaya kalkarsak , hem desen karmaşıklaşır hem de uzar. Bunlar yerine biz [ABCDEF] gibi tanımlayadabilir | karakterinden kısarak fakat bununla kalmayıp bize [A-Z] , [A-F] ,[F-Z] gibi tanımlayabilme şansı da vermişler. Aynıısı rakamlar için de geçerlidir. [0-9] [0-5] gibi. Şimdi aynı metinde belirlediğimiz karakter sınıfından elemanlar arayarak deneyelim.

```
$seslesti=preg_match_all('/[A-Za-z]/',$metin,$seslesme);
if ($seslesti)
print_r($seslesme);
```

5

Bu eşleşmeleri yazdırmaya kalkarsak 43 elemanlı bir dizi olacaktır. örnek çıktığı vermeden deseni açıklayalım :

“A-Z” aralığında ve ya “a-z” aralığında olan karakteri aradık . _all ile eşleşme bitene kadar tekrar aramasını sağladık. Hem “A-Z” hem de “a-z” kullanmamızın nedeni büyük – küçük harf duyarlılığıydı deseni “/A-Z/i” ya da “/a-z/i” gibi “ignore case modifier” ekleyerek büyük küçük harf duyarlılığını kaldırıp kısaltabilirdik çıktımızın elemanları Türkçe karakterleri de içermez.

Subpattern (desen grupları ve alt desenler)

Regex eşleşmeleri genellikle metni gruplamada kullanılır. Örn : bir url i parçalayarak sadece domainname ayırtırmak gibi. Bu grupların bize tek başına dönmesi regexi kullanmamızdaki en büyük neden diyebilirim. Çünkü birbiriyle bütün oluşturan bu grupları biz tek parça alıp grup grup kullanmak isteriz. Her yazdığımız kural parantez içine alınıp art arda aranır tamamı eşleştiyse bize bu grupların hepsi döner.

```
$seslesti=preg_match_all('([A-Z])([A-Z])/i',$metin,$seslesme);
if ($seslesti)
print_r($seslesme);
```

2 gruplu kuralımızda ilk grupta A an Z ye bir karakter diğer grupta ise A-Z dışında kalan 1 karakter. Aradık. Eşleşmemize uyan 11 kısım var _all fonksiyonu gruplarla kullanılırken bize her grup için 1 dizi ayrıca tam eşleşme için de 1 dizi dönderir ilk dizinin elemanları her iki grubun eşleşmesiyle oluşan tam karakterleri verir . her grup için de biz dizi döner. “a” eşleşmesini düşünürsek eğer biri A-Z sınıfına girer diğeri Türkçe karakter olduğu için bu sınıfa girmez dolayısıyla aradığımız şey budur. Ve 3 elemanlı dizin 1 elemanı “a” diğerleri de “a” ve “ı” içerir.

Kaçış Karakteri “ \ ”

Regexte de diğer programlama dillerinde olduğu gibi işaretçileri özel karakterleri kullanıyoruz dedik. Peki bu karakterlerin birini metinde aramaya kalkarsak ne yaparız ? Kaçış karakteri dediğimiz karakterler hemen her dilde bize yardımcı olduğu gibi regexte de işe yarıyor. Bir metin içinde parantez aradığımızı düşünelim ? desenimiz : “ (+ “ en az 1 parantez açma işaretinin olduğu bir parça aramaya kalkarsak bu desenle hata alırız. Preg_last_error(); fonksiyonu ile son hatayı aldığımızda bize “*Compilation failed: nothing to repeat at offset 1*” benzeri bir hata döner. desen indisinin sıfırdan başladığını soylersen 1 bizim için 2 inci karakter oluyor. İkinci karakter için tekrar edecek bir şey bulamadığını soyluyor. Çünkü parantez açma işareti “ (” subpattern başlangıcı anlamına geliyor . şimdi escape edip deneyelim

Yeni desenimiz : “ \ (+ “ bu bize hatasız olarak parantezleri bulabilecektir. Aradığımız şey uzunsa ve bir desenin içine yerleştireceksek bunu preg_quote fonksiyonuna yaptırmamız daha mantıklı olur

Örnek desen : “([“.preg_quote(“ . \ + * ? [^] \$ () { } = ! < > | : - ”.])” bu desen bize regex özel karakterlerini bulabilecektir. Preg_quote bizim için tüm özel karakterlerin başına “\” karakterini ekler.

Bu bölümde diğer tüm niteleyicileri bir arada kullanarak hem göz hem el alışkanlığı yapmaya çalışacağız. İlk örneklerimiz niceleyici karakterlerle ilgili olacaktır. diğer desenlerden örnek verecek olursak A-Z arası bir karakter aramada kullandığımız [A-Z] en az 2 karakter arıyorsak ? ya da bu karakterlerden ard arda sadece 5 tane içeriyorsa eşleştirmek istiyorsak tekrar belirteçleri yardımcımız olacaktır.

Tüm örnekleri üzerinde işleyeceğimiz metni yazalım :

```
$metin="Bugün 19 haziran 2010 cumartesi. \r\n Saat 11:25'te toplantımız var sanırım bu katılacağım 5inci toplantı. \r\nŞuan iş yerinde personel ekranında 26 yazılımcının şirkette olduğu görünüyor."
```

Metinde geçen sayıları bulalım ilk önce. En az 2 rakam içerenleri bulmaya çalışalım `"/[0-9]{2,}/m"` [0-9]arası rakamların yan yana gelerek en az 2 haneli olacak şekilde oluşturdukları sayıları aradık. Modifierlar arasında sadece "m" var rakamlar çok satırlı bir metinde her zaman için bu modifiyeri kullanalım.(özel durumla hariç). Şimdi eşleşmeyi gerçekleştirecek kodumuzu yazalım :

```
$seslesti=preg_match('/[0-9]{2,}/m',$metin,$seslesme);
```

Örnek çıktıya bakalım şimdi :

```
Array ( [0] => Array (
    [0] => 19
    [1] => 2010
    [2] => 11
    [3] => 25
    [4] => 26  ));
```

4 elemanımız da bulduk şimdi alamadığımız sayılara bakalım var mı? "bu katılacağım 5inci toplandı" kısmındaki 5'i alamadık çünkü kriterlerimizde minimum 2 rakamdan oluşması vardı.

Metinden saat formatındakileri bulmaya çalışalım şimdilik
İlk önce kriterlerimizi yazıp sonra bunu bir desene dönüştürelim

- (iki rakam)(iki nokta üst üste)(iki rakam)
ilk eşleşmemiz bize bu metin için doğru sonuç verse bile (birinin boyu 1.80 olan 2 insan arasından 1.80 boyu olan insanı aramak gibi) bu formata uyan başka metinlerde yardımcı olamayacaktır kriterleri katılaştırmamız lazım elemeleri arttırmamız lazım
- (iki rakam) kısmını katılaştıralım ilk rakam sadece 0,1 ve 2 olabilir. İkinci rakam da 0-9 olabilir.fakat ilk rakamımız 2 ise bu en fazla 4 olabilir. (00 ve 24)
- (iki nokta üst üste) bu kısım için yapacağımız bir değişiklik olmaz çünkü iki nokta zaten kesin bir tanım
- (iki rakam) en sonraki iki rakam dakikayı vereceği için 00 ile 59 arasında olabilir. Yani ilk rakamımız 0-5 diğeri 0-9 başka bir şart belirlememize gerek yok.

Kriterlerimizi yazmadan desene dökmek acı sonuçlara yol açabilir. Sonradan uğraşmamak için ilk etapta bunu yazmamız bizim için büyük avantaj olacaktır. şimdi ilk kısmın desenini hazırlayalım:

(iki rakam) saat hanesi :

`(([01][0-9])|([2][0-4]))` 2 alternatifimiz vardı bunu “ | ” ile ayırdık. Ya 0-1 ile 0-9 dan biri yan yana olacak ya da 2 ile 0-4 ten biri. Minimum 00 maximum 24 ü elde edebildik.

(iki nokta) ayırcaç:

`(\.)` kesin bir ifademiz var iki nokta üst üste. Preg_quote fonksiyonunu açıklarken değindiğimiz özel karakterlerden biri olduğu için bunu escape karakteri ile pasif ettik artık özel bir karakter değil bildimiz iki nokta üst üste işlevini görecekler

(iki rakam) dakika hanesi:

`([0-5][0-9])` bu ca çok basit bir ifade oldu. 0-5 ile 0-9 dan ikisi yan yana duracak bu kadar basit.

Şimdi saati bulabilecek desene bakalım:

`"(((01)[0-9])|((2)[0-4]))(\.)([0-5][0-9])"` dikkat edecek olursak parantezlerimiz arttı.

Çünkü `"((01)[0-9])|([2][0-4])"` bu ifade bir bütün ve içerdekiler birbirinin alternatifi olduğu için bunları tek paranteze aldık. PHP kodumuzu da yazalım

```
$seslesti=preg_match('/(((01)[0-9])|((2)[0-4]))(\.)([0-5][0-9])/m',$metin,$seslesme);
```

Yine mutiline modifiyerdan vazgeçmedik. Bu desende işimize yaramadı ama \$seslesme dizisini kontrol edecek olursak. Minimum 5 elemanlı bir dizi olur.

İlk eleman eşleşmenin tamamı diğer elemanlar da subpatternlar, yani parantez içleri eşleşmeleri.

Örnek çıktığı vermeden önce bir de isimlendirelim

```
$seslesti=preg_match('/(?<saat>((01)[0-9])|((2)[0-4]))(\.)(?<dakika>[0-5][0-9])/m',$metin,$seslesme);
```

saat ve dakikayı içeren desenlere isim verdik ve çıktımıza bakıyoruz.

<pre>Array ([0] => 11:25 [saat] => 11 [1] => 11 [2] => 11 [3] => [4] => : [dakika] => 25 [5] => 25)</pre>	<p>Yine her zamanki gibi ilk elemanımız eşleşmenin tamamı. Diğer elemanlar da alt eşleşmeler. Burada dikkat edeceğimiz bir şey daha var isimlendirdiğimiz alt eşleşmeler hem indisleriyle hem isimleriyle geldiler. Yani isim kullandığımız her eşleşme için 2 eleman oluşacaktır. Bu işlemden sonra gönül rahatlığı ile</p> <pre>\$seslesme['saat']; \$seslesme['dakika']; kullanabileceğiz.</pre> <p>Ne kadar keyifli değil mi?</p>
---	---