عبارات منظم

قالب کلی یک عبارت منظم به شکل زیر است.

/pattern/flags

نکته : در برخی مواقع ممکن است از کاراکترهای دیگری به جای "/" به عنوان جداکننده‌ی بخش pattern و flags استفاده شود.

pattern : قسمت اصلی عبارت منظم را تشکیل می‌دهد و نشان دهنده‌ی عبارت مورد جستجو است.

flags : در این قسمت می‌توان روش جستجو را کنترل کرد. مثلا استفاده از فلگ i باعث می‌شود جستجو به صورت Case insensitive انجام شود. همچنین استفاده از فلگ g باعث می‌شود با یافتن اولین مورد تطابق، جستجو متوقف نشده و تا پایان رشته ادامه یابد.

مثال : جستجوی کلمه regex بدون توجه به حالت حروف.

/regex/i

کاراکترهای خاص (Meta characters) : برخی کاراکترها در عبارات منظم دارای معنی خاصی هستند. با استفاده از این کاراکترها می‌توان انواع پیچیده‌ای از عبارات منظم را ایجاد کرد. این کاراکترها عبارتند از :

. \ [ ] ? + * { } | ( ) ^ $ /

کاراکتر نقطه (Wildcard) : این کاراکتر می‌تواند جایگزین هر کاراکتری به جز خط جدید (new line) شود. (در حالت پیش‌فرض)

مثال : کلمات سه حرفی که با حرف "c" شروع می‌شوند و به حرف "t" ختم می‌شوند. (این سه حرف ممکن است بخشی از یک رشته بزرگتر باشند.)

/c.t/i ➔ cut , cat , cutter , Category

نکته : در صورتی که بخواهیم یکی از کاراکترهای خاص را بدون معنی خاصشان داخل عبارات منظم به کار ببریم، باید از کاراکتر "\" قبل از کاراکتر مورد نظر استفاده کنیم.

/c.t\./ ➔ cat. , cute

مجموعه‌ها ([ ]) : فقط یکی از کاراکترهای داخل یک مجموعه می‌تواند با یک کاراکتر از رشته‌ی مورد نظر مطابقت داشته باشد.

[abc] ➔ یکی از حروف داخل براکت

/[cr]at/ ➔ cat , rat , bat

نکته : داخل مجموعه‌ها می‌توان از بازه‌های کاراکتری استفاده کرد.

[0-4] ➔ [01234]

[c-g] ➔ [cdefg]

[A-D] ➔ [ABCD]

[a-zA-Z0-9] ➔ تمام حروف الفبای انگلیسی و اعداد

[ی-م] ➔ [منوهی]

نکته : داخل مجموعه‌ها فقط چهار کاراکتر "\"، "^"، "]" و "-"، کاراکتر خاص محسوب می‌شوند.

نکته : با قرار دادن کاراکتر "^" در ابتدای مجموعه، می‌توان کاراکترهای یک مجموعه را معکوس کرد.

[^0-9] ➔ هر کاراکتری به جز اعداد

[^a-z] ➔ هر کاراکتری به جز حروف کوچک الفبای انگلیسی

[^.] ➔ هر کاراکتری به جز نقطه

نکته : کاراکتر "^" فقط در صورتی که در ابتدای مجموعه باشد دارای معنی خاص است. در غیر این صورت معنی خاصی ندارد. همچنین کاراکتر "-" اگر در انتهای مجموعه باشد معنی خاصی ندارد.

مجموعه‌های پرکاربرد : جهت خلاصه نویسی برخی مجموعه‌های کاراکتری که کاربرد زیادی در عبارات منظم دارند از برخی نمادهای خاص استفاده می‌شود.

نماد	معادل	توضیح
\w	[a-zA-Z0-9_]	کاراکترهای الفبایی (انگلیسی)
\W	[^a-zA-Z0-9_]	کاراکترهای غیر الفبایی (انگلیسی)
\s	[\t\n\r]	کاراکترهای فضای خالی
\S	[^\s]	تمام کاراکترها به جز فضاهای خالی
\d	[0-9]	کاراکترهای عددی (انگلیسی)
\D	[^0-9]	کاراکترهای غیر عددی (انگلیسی)

نکته : این مجموعه‌ها را می‌توان داخل مجموعه‌های دیگر به کار برد.

/[\da-z]/ig ➔ تمام حروف الفبای انگلیسی و اعداد

شمارنده‌ها (Quantifiers) : جهت مشخص کردن تعداد تکرار یک یا چند کاراکتر در عبارات منظم از شمارنده‌ها استفاده می‌شود.

Quantifier	معنی	مثال
?	صفر یا یک تکرار	/colou?r/ ➔ color , colour , colorize
+	یک تکرار یا بیشتر	/1+3/ ➔ 3 , 13 , 113 , 1113 , 21134
*	صفر تکرار یا بیشتر	/1*3/ ➔ 3 , 13 , 113 , 1113 , 21134
{n}	دقیقا n تکرار	/1{2}3/ ➔ 3 , 13 , 113 , 1113 , 21134
{n,}	n تکرار یا بیشتر	/1{2,}3/ ➔ 3 , 13 , 113 , 1113 , 21134
{n,m}	بین n تا m تکرار	/1{1,2}3/ ➔ 3 , 13 , 113 , 1113 , 21134

نکته : در صورت استفاده از فلگ s، کاراکتر نقطه می‌تواند با خط جدید هم مطابقت کند.

/<.+>/s ➔ This is my picture.

رفتار Greedy و Lazy در شمارنده‌ها : در حالت پیش‌فرض اگر زیر رشته‌های متفاوتی با شمارنده‌ها مطابقت داشته باشند، بلندترین مطابقت انتخاب می‌شود. این حالت را Greedy می‌نامند.

/<.+>/g ➔ do not press the button.

/H.+d/g ➔ Hamid and Hamed

نکته : با قرار دادن کاراکتر "?" بعد از Quantifier ها، می‌توان رفتار آنها را به حالت Lazy تغییر داد. در این حالت کوتاه‌ترین مطابقت انتخاب خواهد شد.

/<.+?>/g ➔ do not press the button.

/H.+?d/g ➔ Hamid and Hamed

/1{2,}?3/g ➔ 11131111311

گروه‌ها (Parenthesis) : با استفاده از پرانتز می‌توان بخشی از یک الگو را به عنوان یک گروه تعیین کرد. گروه‌ها کاربردهای مختلفی دارند که به مرور به بررسی آنها می‌پردازیم.

اعمال شمارنده‌ها به چند کاراکتر :

/a(bc)*d/ ➔ ad , abcd , abbcbcd , abcbcbcd , abcabc

/a(bc){2,}d/ ➔ ad , abcd , abbcbcd , abcbcbcd , abcabc

انتخاب بین چند الگو (|) : با استفاده از کاراکتر "|" می‌توان چند الگو را با یکدیگر OR کرد. این کار معمولا داخل گروه‌ها انجام می‌شود.

/b|car/i ➔ car , bar , Barcelona , oscar

/(Sun|Mon|Tue|Wed|Thu|Fri|Sat) \d{1,2}/ ➔ Sun 18 , Mon 197

نکته : در انتخاب بین الگوها، جستجو از سمت چپ آغاز می‌شود. و با کشف اولین مطابقت، جستجو خاتمه می‌یابد.

/\d{2}|\d{4}/ ➔ 1234

/\d{4}|\d{2}/ ➔ 1234

نکته : جهت جلوگیری از بروز ابهام و همچنین خلاصه‌نویسی، بهتر است بخش‌هایی که دارای اشتراک هستند را با هم ترکیب کنیم.

/cutter|cut|butter|but/ ➔ /cut(ter)?|but(ter)?/

/\d{4}|\d{2}/ ➔ /\d{2}(\d{2})?/

تعیین موقعیت با Anchor ها : گاهی اوقات لازم است علاوه بر درست بودن ترتیب کاراکترها، موقعیت آنها در رشته نیز بررسی شود. در این موارد می‌توان از Anchor ها استفاده کرد.

شروع و پایان رشته (^ و $) :

/^set/ ➔ set , my set , reset , settle

/set$/ ➔ set , my set , reset , settle

/^set$/ ➔ set , my set , reset , settle

/^<.+>$/ ➔ <hr /> , he is a student , strong

نکته : در صورت استفاده از فلگ m، ابتدا و انتهای هر سطر از رشته، با Anchor های ^ و $ مطابقت خواهند کرد.

نکته : "\A" و "\Z" حتی با فعال بودن فلگ m، باز هم فقط با ابتدا و انتهای کل رشته مطابقت خواهند کرد. (این موارد کاربرد زیادی ندارند)

شروع و پایان کلمات (Word Boundaries) :

- **\b** : فضای بین یک کاراکتر از \w و یک کاراکتر از \W
- **\B** : فضای بین دو کاراکتر از \w یا دو کاراکتر از \W (کاربرد زیادی ندارد)

/\bset/ ➔ set , my set , reset , settle , 2set , @set

/\bset\b/ ➔ set , my set , reset , settle , 2set , @set

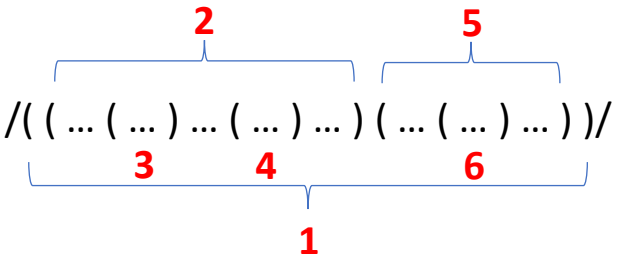
/\Bset/ ➔ set , my set , reset , settle , 2set , @set

نکته : ابتدا و انتهای رشته (^ و $) جزئی از \W به حساب می‌آیند.

گروه‌های ضبط شونده (Capturing Groups) : در حالت پیش‌فرض تمام گروه‌های موجود در عبارات منظم ضبط (Capture) می‌شوند. گروه‌های ضبط شده به ترتیب ظاهر شدنشان در عبارات منظم شماره گذاری می‌شوند.

```
   1        2    3
/(<h3>)\d+-(.+)(</h3>)/g
```

نکته : در صورت استفاده از پرانتزهای تو در تو، ابتدا پرانتز خارجی، سپس پرانتزهای داخلی آن، سپس پرانتزهای بعدی شماره گذاری می‌شوند.



Backreference ها : با استفاده از Backreference ها می‌توان گروه‌های ضبط شده را مجددا در الگو به کار برد.

/<(.+)>.*<\/\1>/ ➔ hello , hello</i>

نکته : Backreference ها را با هر ترتیبی می‌توان به کار برد. همچنین می‌توان از یک گروه ضبط شده چند بار استفاده کرد.

/(ab)(cd)\2\1\2/ ➔ abcdcdabcdcd

نکته : در گروه‌هایی که همراه با شمارنده‌ها به کار می‌روند، فقط آخرین مطابقت ضبط می‌شود.

/([acr]+)=\1/ ➔ car=car , car=r

/([acr])+=\1/ ➔ car=car , car=r

نکته : Backreference ها را نمی‌توان داخل مجموعه‌ها (براکت) به کار برد.

گروه‌های نامگذاری شده (Named Group) : جهت ساده‌تر شدن استفاده از Backreference ها و خوانایی بیشتر عبارات منظم، می‌توان برای گروه‌ها یک نام دلخواه انتخاب کرد.

/<(?<tag>.+)>.*<\/\k<tag>>/ ➔ hello , hello</i>

نکته : روش فوق در بیشتر زبان‌های برنامه‌نویسی امروزی مانند PHP، Java، JavaScript، C# و ... قابل استفاده است. اما در برخی زبان‌ها از روش‌های دیگری برای این منظور استفاده می‌شود. مثلا در Python از این روش استفاده می‌شود.

/<(?P<tag>.+)>.*<\/?P=tag>/

جلوگیری از ضبط شدن گروه‌ها (Non-Capturing Groups) : در صورتی که نیازی به استفاده از یک گروه به عنوان Backreference نداشته باشم. می‌توان با قرار دادن علامت "?:" در ابتدای گروه، آن را به یک Non-Capturing Group تبدیل کرد.

/(?:ab)(cd)\1/ ➔ abcdcdab

نکته : استفاده از Non-Capturing Group ها باعث افزایش سرعت اجرا، کاهش مصرف حافظه و همچنین امکان استفاده از گروه‌های بیشتر می‌شود. (در برخی محیط‌ها فقط می‌توان از \1 تا \9 را به عنوان Backreference استفاده کرد.)

جایگزینی در عبارات منظم : با استفاده از عبارات منظم می‌توان بخش‌های مطابقت یافته از متن اصلی را، با متن دلخواه جایگزین و یا کاملا حذف کرد.

/abc/ , {} ➔ 123abc123 ➔ 123123

/abc/ , {cba} ➔ 123abc123 ➔ 123cba123

نکته : علامت آکلاد در مثال‌های فوق هیچ معنی خاصی ندارد و صرفا برای نمایش مقدار جایگزین شونده استفاده شده است.

استفاده از گروه‌های ضبط شده : می‌توان از گروه‌های ضبط شده در جایگزینی استفاده کرد. برای این منظور باید از کاراکتر $ به همراه شماره گروه ضبط شده استفاده کرد.

```
/<(h[1-6])(.*?)>(.*?)<\/\1>/ , {<$1$2 class="myclass">$3<\/$1>}
<h1 id="title">Hello</h1>
```

➔ <h1 id="title" class="myclass">Hello</h1>

نکته : در برخی زبان‌های برنامه نویسی (مانند PHP) علاوه بر $ می‌توان از \ هم برای اشاره به گروه‌های ضبط شده استفاده کرد. در برخی زبان‌ها (مانند JavaScript) فقط می‌توان از $ استفاده کرد. و در برخی زبان‌ها (مانند Python) فقط می‌توان از \ استفاده کرد.

نکته : در برخی زبان‌ها (مانند JavaScript) می‌توان از گروه‌های نامگذاری شده در زمان جایگزینی استفاده کرد.

```
/<(?<tag>b)>(.*?)<\/\k<tag>>/ , {<$<tag>>Hello $2</$<tag>>}
```

بررسی شرایط با استفاده از Assertion ها (Lookaround)

گاهی اوقات نیاز است تا قبل از بررسی مطابقت یک رشته با یک الگوی مشخص، بررسی کنیم که آیا رشته مورد نظر شرایط خاصی را دارد یا خیر؟ در چنین شرایطی می‌توان از Assertion ها استفاده کرد. Assertion ها دارای ۴ نوع مختلف مطابق جدول زیر هستند.

نوع Assertion	نحوه استفاده
Positive Lookahead	(?=subpattern)
Negative Lookahead	(?!subpattern)
Positive Lookbehind	(?<=subpattern)
Negative Lookbehind	(?<!subpattern)

در ادامه به معرفی هر یک از انواع Assertion ها می‌پردازیم.

نکته : Assertion ها نیز مانند Anchor ها، هیچ کاراکتری را انتخاب نمی‌کنند. فقط برقرار بودن شرایط را بررسی می‌کنند.

Positive Lookahead : با استفاده از این Assertion می‌توان بررسی کرد که آیا شرایط خاصی در **ادامه رشته** برقرار است یا خیر؟ در صورت مثبت بودن پاسخ، مطابقت کاراکترها با عبارت منظم می‌تواند ادامه پیدا کند.

/(?=.*board)black/ ➔ blackboard , blacksea , onboard

/\w+(?=,)/g ➔ "C, C++, PHP, Java and JavaScript"

نکته : Assertion ها معمولا روی رشته‌های کوتاه (تک خطی) به کار می‌روند.

نکته : پرانتزهایی که برای تعریف Assersion ها به کار می‌روند ضبط نمی‌شوند. اما پرانتزهای داخل آنها ضبط می‌شوند.

/\w+(?=(,))\1/ ➔ "C, C++, PHP, Java and JavaScript"

مثال : عبارت منظمی که فقط با رشته‌ای مطابقت می‌کند که حداقل ۸ و حداکثر ۱۵ کاراکتر باشد. همچنین شامل فضای خالی نباشد، و از هر یک از مجموعه‌های [a-z]، [A-Z]، [0-9] و [^\w] حداقل یک کاراکتر در آن موجود باشد.

/^(?=.*[a-z])(?=.*[A-Z])(?=.*[0-9])(?=.*[^\w\s])\S{8,15}$/

➔ A0aaaaaaaaa , A0aaaaaaa#aa , A0aaaaسaaaa

Negative Lookahead : در این نوع Assertion نیز بررسی می‌شود که آیا شرایط خاصی در ادامه‌ی رشته برقرار است یا خیر؟ و در صورت برقرار منفی بودن پاسخ، مطابقت کاراکترها با عبارت منظم می‌تواند ادامه پیدا کند.

/(?!.*\.\.)[\w.]+/ ➔ ab.cd.e , ab..cd , .abcd.efg.

Positive Lookbehind : در این نوع Assertion بررسی می‌شود که آیا شرایط خاصی در بخش‌های قبلی رشته (قبل از Assertion) برقرار است یا خیر؟ در صورت مثبت بودن پاسخ، مطابقت کاراکترها با عبارت منظم می‌تواند ادامه پیدا کند.

/(?<=<(b|strong)>).*?(?=<\/\1>)/ ➔ Hello

Negative Lookbehind : در این نوع Assertion بررسی می‌شود که آیا شرایط خاصی در بخش‌های قبلی رشته (قبل از Assertion) برقرار است یا خیر؟ در صورت منفی بودن پاسخ، مطابقت کاراکترها با عبارت منظم می‌تواند ادامه پیدا کند.

/(?<!the)\b([a-zA-Z])+/g ➔ In the name of God.

نکته : Positive Lookbehind و Negative Lookbehind در بیشتر محیط‌ها و زبان‌های برنامه‌نویسی به طور کامل پشتیبانی نمی‌شوند. (در ECMAScript 2018 به بعد کاملا پشتیبانی می‌شوند.)

عبارات منظم در HTML

در HTML می‌توان از صفت pattern برای تگ‌های <input> استفاده کرد و یک عبارت منظم را تعیین کرد. در این صورت فقط در صورتی که متن وارد شده توسط کاربر با عبارت منظم مطابقت داشته باشد، امکان ارسال فرم توسط کاربر وجود دارد.

```
<input type="text" pattern="\d{3,6}" />
```

نکته : در HTML کل رشته وارد شده باید با pattern مطابقت داشته باشد. یعنی به صورت خودکار Anchor های ^ و $ در ابتدا و انتهای pattern اضافه می‌شوند.

نکته : در HTML نمی‌توان از فلگ‌ها استفاده کرد. البته معمولا نیازی به آنها نیست. اما در صورت نیاز به فلگ i، می‌توان از (?i) در ابتدای pattern استفاده کرد.

عبارات منظم در JavaScript

در جاوا اسکریپت به دو روش می‌توان عبارات منظم را تعریف کرد.

```
let regex1 = /^\d+$/gm
let regex2 = new RegExp('^\\d+$' , 'gm')
```

متد test : با استفاده از این متد می‌توان بررسی کردن که آیا یک عبارت منظم با یک رشته‌ی خاص مطابقت دارد یا خیر؟

```
regex1.test("4432")     ➔   true
regex1.test("Hello")    ➔   false
```

متد exec : این متد علاوه بر بررسی کردن مطابقت یا عدم مطابقت یک رشته با یک عبارت منظم، می‌تواند محل دقیق مطابقت و رشته‌ی مطابقت یافته را نیز مشخص کند.

```
regex1.exec("4432")     ➔   ["4432", index: 0, input: "4432"]
```

متد replace : با استفاده از این متد نیز می‌توان جایگزینی رشته‌ها را هم به صورت عادی و هم با استفاده از عبارات منظم انجام داد.

```
let str = "Hello World!"
str = str.replace(/[a-z]+/gi, "x")      ➔    x x!
```

عبارات منظم در PHP

تابع preg_match : با استفاده از این تابع می‌توان یک رشته را جهت مطابقت با یک عبارت منظم بررسی کرد.

```
preg_match("/[A-Z]([a-z]{2,})/" , "Hello my friend" , $matches);
```

در صورت وجود الگوی تعیین شده در رشته‌ی مورد نظر، این تابع مقدار 1، و در غیر این صورت مقدار 0 را بازمی‌گرداند.

نکته : بخشی از رشته که با الگو مطابقت یافته است و همچنین بخش‌های ضبط شده در متغیر $matches به صورت یک آرایه ذخیره می‌شوند. نتیجه مثال فوق به صورت زیر خواهد بود.

```
$matches[0] = "Hello"
$matches[1] = "ello"
```

نکته : در PHP امکان استفاده از فلگ g در عبارات منظم وجود ندارد. اما می‌توان از تابع preg_match_all به جای تابع preg_match استفاده کرد. استفاده از این تابع به معنی فعال کردن فلگ g است. در این حالت مقداری که در متغیر $matches قرار می‌گیرد، یک آرایه دو بعدی است. که اندیس اول کل موارد مطابقت یافته، و اندیس‌های بعدی موارد ضبط شده را نگهداری می‌کنند.

تابع preg_replace : با استفاده از این تابع می‌توان عمل جایگزینی را در عبارات منظم انجام داد.

```
preg_replace('/<(b)>(.*?)<\/\1>/i' , '<strong>$2</strong>' , $text);
```

نکته : در PHP بهتر است برای تعریف عبارات منظم از Single Quotation استفاده کنیم. در صورت استفاده از Double Quotation باید به ازای ار "\" از دو "\" استفاده کنیم.

```
preg_replace("/<(b)>(.*?)<\\/\\1>/i" , '<strong>$2</strong>' , $text);
```