رجکس

در این بخش میخواهیم با رجکس و استفاده از آن در پایتون آشنا شویم!

رجکس (*Regular Expression* یا به اختصار *RegEx*) یک الگو یا قالب است که برای جست‌وجو و کار با متن‌ها استفاده می‌شود. با استفاده از رجکس، می‌توانیم الگوهای خاصی از رشته‌ها (مثل کلمات، اعداد، یا ترکیبی از حروف) را پیدا کنیم، جایگزین کنیم، یا آن‌ها را بررسی کنیم. این الگوها، خود از جنس رشته هستند. مثلاً می‌توانیم با یک رجکس ساده، تمام ایمیل‌ها یا شماره‌های تلفن موجود در یک متن را پیدا کنیم. شایان ذکر است که رجکس مفهوم مستقلی است و فقط مخصوص پایتون نیست، بلکه در همه‌ی زبان‌های برنامه‌نویسی وجود دارد.

به‌عنوان مثال، برای ثبت‌نام در بسیاری از سایت‌ها لازم است آدرس ایمیل خود را وارد کنیم. برای بررسی کردن این‌که آیا این آدرس ایمیل معتبر است و با الگوی آدرس‌های ایمیل مطابقت دارد، از رجکس استفاده می‌شود. یا مثلاً ممکن است در برنامه‌ای نیاز داشته باشیم پلاک ماشین کاربر را از او ورودی بگیریم. در این صورت، ورودی او را با یک الگوی رجکس مقایسه می‌کنیم تا از معتبر بودن آن مطمئن باشیم و اگر کاربر اطلاعات غلطی وارد کرد، برنامه‌ی ما به مشکل نخورد. ساختن الگو با رجکس بسیار ساده است. تقریباً همه‌ی حروف و اعداد به جای خودشان در الگو هستند. مثلاً اگر بنویسیم `abc` ، هر رشته‌ای که این سه حرف را پشت سر هم داشته باشد، با الگوی ما مطابقت دارد. اما بعضی علامت‌ها، کاربردهای خاصی در رجکس دارند. در زیر به معرفی این علامت‌ها می‌پردازیم:

علائم معنادار

علامت `.`

علامت `.` در رجکس به معنای هر کاراکتری به‌جز خط جدید است. به عبارت دیگر، این علامت می‌تواند جایگزین هر حرف، عدد، یا نشانه‌ای در یک متن شود. برای مثال رشته‌های زیر با الگوی `....` مطابقت دارند:

```
"abcd"
"$^%#"

```

```
"1234"
```

علامت \

علامت \ در رجکس برای فرار (Escape) دادن علامت‌های خاص استفاده می‌شود تا بتوان آن‌ها را به جای خودشان به کار برد. مثلا اگر بخواهیم از علامت . استفاده کنیم به طوری که فقط با کاراکتر . مطابقت کند و با باقی کاراکترها مطابقت نداشته باشد، یک \ پیش از آن قرار می‌دهیم. گاهی این علامت به حروف خاص نیز معنای دیگری می‌بخشد، که جلوتر به آن پرداخته شده است.

برای مثال در نمونه‌ی زیر، الگوی .\... با دو رشته‌ی اول مطابقت دارد ولی با آخری مطابقت ندارد، چون در انتهایش . ندارد:

```
"abc."
"123."
"defg"
```

علامت \d

این علامت با همه‌ی ارقام (کاراکترهای 0 تا 9) مطابقت دارد. به‌عنوان مثال، الگوی را در نظر بگیرید. این الگو با دو رشته‌ی اول مطابقت دارد، ولی با رشته‌ی آخر مطابقت ندارد، چون کاراکتر آخر آن رقم **نیست**:

```
"asd7"
"6sd1"
"dsdg"
```

علامت \D

این علامت برعکس علامت بالا است و با همه‌ی کاراکترها به جز ارقام (کاراکترهای 0 تا 9) مطابقت دارد. به‌عنوان مثال، الگوی .\D را در نظر بگیرید. این الگو با دو رشته‌ی اول مطابقت دارد، ولی با رشته‌ی آخر مطابقت ندارد، چون کاراکتر آخر آن رقم **است**:

```
"bQ"
"6s"
"d5"
```

علامت \w

این علامت با همه‌ی حروف (کوچک و بزرگ) انگلیسی، ارقام (کاراکترهای 0 تا 9) و کاراکتر _ مطابقت دارد. برای مثال در نمونه زیر، الگوی \w\w با دو رشته‌ی اول مطابقت دارد، ولی با آخری مطابقت ندارد:

```
"aB"
"6_"
"<?"
```

علامت \W

این علامت برعکس علامت بالا است و با همه‌ی کاراکترها به‌جز حروف (کوچک و بزرگ) انگلیسی، ارقام (کاراکترهای 0 تا 9) و کاراکتر _ مطابقت دارد. برای مثال در نمونه زیر، الگوی \W.. با دو رشته‌ی اول مطابقت دارد ولی با آخری مطابقت ندارد، چون در انتهایش کاراکتر عددی **دارد**:

```
"bQ@"
"6_!"
"d56"
```

علائم نمایانگر تعداد

علامت +

این علامت نشان‌دهنده‌ی این است که کاراکتر قبل از آن باید حداقل یک بار (یک بار یا بیشتر) **در همان موقعیت و پشت سر هم** تکرار شده باشد و می‌تواند با باقی علامت‌های گفته‌شده هم ترکیب شود. برای مثال، الگوی sx+ با دو رشته‌ی اول مطابقت دارد ولی با دو رشته‌ی آخر مطابقت ندارد:

```
"sx"
"sxxx"
"s"
"xs"
```

علامت *

این علامت نشان‌دهنده‌ی این است که کاراکتر قبل از آن باید **صفر** بار یا بیشتر **در همان موقعیت و پشت سر هم** تکرار شده باشد و می‌تواند با باقی علامت‌های گفته‌شده هم ترکیب شود. برای مثال الگوی `sx*` با سه رشته‌ی اول مطابقت دارد، ولی با رشته‌ی آخر مطابقت ندارد:

```
"sx"
"sxxx"
"s"
"xs"
```

علامت ?

این علامت نشان‌دهنده‌ی این است که کاراکتر قبل از آن باید حداکثر یک بار (صفر بار یا یک بار) **در همان موقعیت** آمده باشد؛ یعنی این کاراکتر می‌تواند در رشته باشد یا نباشد. این علامت هم می‌تواند با باقی علامت‌های گفته‌شده ترکیب شود. برای مثال، در نمونه زیر، الگوی با چهار رشته‌ی اول مطابقت دارد، ولی با رشته‌ی آخر مطابقت ندارد، چون کاراکتر `e` دو بار تکرار شده‌است:

```
"df"
"ef"
"f"
"def"
"deef"
```

علامت {}

هنگامی که یک کاراکتر باید به تعداد مشخصی در رشته‌مان تکرار شده باشد، از این علامت استفاده می‌کنیم. مثلاً `a{23}` یعنی حرف `a` ۲۳ بار پشت سر هم تکرار شده باشد، یا `g{4,9}` یعنی حرف `g` بین ۴ تا ۹ بار پشت سر هم تکرار شده باشد. هم‌چنین این علامت می‌تواند با همه‌ی علامت‌های گفته‌شده نیز ترکیب شود و به معنی تکرار الگوهای مشخص‌شده توسط آن علامت‌ها به تعدادی مشخص است. برای مثال در نمونه زیر، الگوی `.{3,8}` با سه رشته اول مطابقت دارد ولی با آخری مطابقت ندارد، چون طولش بین ۳ و ۸ نیست:

```
"@!#$"
"IAmAli"
"12345678"
"Hi"
```

علائم انتخاب و دسته‌بندی

علامت `[]`

گاهی اوقات می‌خواهیم یک کاراکتر در رشته‌ی ما از بین کاراکترهای خاصی باشد. در این موارد این گروه از کاراکترها را در بین علامت `[]` قرار می‌دهیم. دقت کنید این علامت فقط با یک حرف *Match* می‌شود و اگر یک کاراکتر با کاراکترهای درون براکت مطابقت داشته باشد، برنامه به سراغ ادامه‌ی الگو در خارج از براکت می‌رود.

برای مثال در نمونه‌ی زیر، الگوی `[abc][def][ghi]` با دو رشته‌ی اول مطابقت دارد، ولی با آخری مطابقت ندارد، چون کاراکترهای دوم و سوم به دسته‌های مشخص‌شده تعلق ندارند:

```
"beg"
"cdi"
"abc"
```

هم‌چنین برای مشخص کردن گروهی از کاراکترها که در یک بازه قرار دارند، کاراکتر شروع و کاراکتر پایان را با `-` جدا می‌کنیم. مثلاً `[H-M]` نشان‌دهنده‌ی تمام حروف بین `H` و `M` ، با احتساب خود این دو حرف است؛ یعنی `[H-M]` معادل `[HIJKLM]` است. هم‌چنین `[2-7]` نشان‌دهنده‌ی تمام ارقام بین `2` و `7` با احتساب خود این دو رقم است؛ یعنی `[2-7]` معادل `[234567]` است.

برای مثال در نمونه‌ی زیر، الگوی `[i-p]` با سه رشته‌ی اول مطابقت دارد، ولی با رشته‌ی آخر مطابقت ندارد، چون `x` بین کاراکترهای `i` و `p` قرار ندارد:

```
"o"
"i"
"p"
"x"
```

همچنین با اضافه کردن کاراکتر `^` به ابتدای علامت `[]` ، متمم الگو (هرچه در آن الگو نیست) در نظر گرفته می‌شود و اگر کاراکتری برابر با کاراکترهای درون این علامت باشد، پذیرفته نمی‌شود. برای مثال در نمونه‌ی زیر، الگوی `abc[^def]` با دو رشته‌ی اول مطابقت ندارد، ولی با آخری مطابقت دارد، چون در انتهایش یکی از کاراکترهای `d` یا `e` یا `f` نیامده‌است:

```
"abcd"
"abcf"
"abck"
```

Python

Copy

علامت ()

علامت `()` برای گروه‌بندی الگوها استفاده می‌شود تا بتوان آن‌ها را به عنوان یک واحد در نظر گرفت، مخصوصاً هنگامی‌که می‌خواهیم از علامت‌های شمارشی برای گروهی از کاراکترها به جای یک کاراکتر استفاده کنیم. برای مثال در نمونه‌ی زیر، الگوی `"(abc)+"` با دو رشته‌ی اول مطابقت دارد، ولی با آخری مطابقت ندارد، چون در آن کاراکتر `d` به کار رفته‌است.

```
"abcabcabc"
"abc"
"abdccccc"
```

علائم شروع و پایان

علامت ^

اگر این علامت ابتدای الگویی به کار رود، به این معنی است که رشته باید با آن الگو شروع شود. مثلاً الگوی `^Mah.+` با دو رشته اول تطابق دارد ولی با رشته‌ی سوم تطابق ندارد، چون با `Mah` شروع نشده‌است.

```
"Mahtab"
"Mahan"
"TabMah"
```

علامت $

اگر این علامت انتهای الگویی به کار رود، به این معنی است که رشته باید با الگو پایان یابد. مثلاً الگوی `.+pour$` با دو رشته‌ی اول تطابق دارد، ولی با رشته‌ی سوم تطابق ندارد، چون با `pour` پایان نیافته‌است.

```
"Alipour"
"Rashidpour"
"pourMansour"
```

در ادامه، جمع‌بندی این علامت‌ها و چندین الگوی پرکاربرد دیگر را نیز در جداولی قرار دادیم:

کاراکترها

علامت	توضیح	مثال	رشته‌ی متناسب
`\d`	کاراکتر عددی (شامل یک رقم (`0-9`))	`file_\d\d`	`file_25`
`\w`	کاراکتر کلمه‌ای (شامل یک کاراکتر عددی (`0-9`) یا حرفی (`a-z` یا `A-Z`) یا زیرخط (`_`))	`\w-\w\w\w`	`A-b_1`
`\s`	کاراکتر فضای خالی (یک کاراکتر فاصله یا تب (`\t`) یا خط جدید `\n`)	`a\sb\sc`	`a b c`

علامت	توضیح	مثال	رشته‌ی متناسب
\D	یک کاراکتر غیرعددی	\D\D\D	ABC
\W	یک کاراکتر غیرکلمه‌ای	\W\W\W\W\W	*-+=)
\S	یک کاراکتر غیر فضای خالی	\S\S\S\S	Yoyo

تعداد

علامت	توضیح	مثال	رشته‌ی متناسب
+	یک بار یا بیشتر (پشت سر هم)	Version \w-\w+	Version A-b1_1
{3}	دقیقا سه بار (پشت سر هم)	\D{3}	ABC
{2,4}	دو تا چهار بار (پشت سر هم)	\d{2,4}	156
{3,}	سه بار یا بیشتر (پشت سر هم)	\w{3,}	regex_tutorial
*	صفر بار یا بیشتر (پشت سر هم)	A*B*C*	AAACC
?	صفر یا یک بار	plurals?	plural

منطق

علامت	توضیح	مثال	رشته‌ی متناسب
`	`	OR	22`
(…)	گروه گیرا *(capturing group)*	A(nt`	`(pple
\1	محتویات گروه ۱	r(\w)g\1x	regex
\2	محتویات گروه ۲	(\d\d)\+(\d\d)=\2\+\1	12+65=65+12

علامت	توضیح	مثال	رشته‌ی متناسب
`(?: … )`	گروه ناگیرا *(non-capturing group)*	`A(?:nt`	`(pple`

گروه ناگیرا، گروهی است که محتوای آن به‌صورت جداگانه قابل‌دریافت نیست.

کلاس کاراکتر

کلاس	توضیح	مثال	رشته‌ی متناسب
`[ … ]`	یکی از کاراکترهای داخل براکت	`[AEIOU]`	`A`
`[ … ]`	یکی از کاراکترهای داخل براکت	`T[ao]p`	`Tap` و `Top`
`-`	بازه	`[a-z]`	`m`
`[x-y]`	یکی از کاراکترهای بازه‌ی `x` تا `y`	`[A-Z]+`	`GREAT`
`[^x]`	یک کاراکتر غیر از `x`	`[^a-z]{3}`	`A1!`
`[^x-y]`	یک کاراکتر غیر از بازه‌ی `x` تا `y`	`[^0-9]+`	`hello`

کاراکترهای بیشتر

کاراکتر	توضیح	مثال	رشته‌ی متناسب
`.`	هر کاراکتری جز خط جدید	`a.c`	`abc`
`.`	هر کاراکتری جز خط جدید	`.*`	`whatever, man.`
`\.`	کاراکتر نقطه	`a\.c`	`a.c`
`\`	برای کاراکترهای خاص استفاده می‌شود.	`\.\*\+\? \$\^\/\\`	`.*+? $^/\`
`\`	برای کاراکترهای خاص استفاده می‌شود.	`\[\{\(\)\}\]`	`[{()}]`

کاراکتر	توضیح	مثال	رشته‌ی متناسب
^	شروع رشته	^abc	abcsomething
$	پایان رشته	abc$	somethingabc

علامت‌های بسیار دیگری در رجکس وجود دارند و ترکیب‌های بسیار زیاد و پیچیده‌ای می‌توان با این علامت‌ها ساخت.

ساختن الگو با رجکس بسیار ساده است. تقریباً همه‌ی حروف و اعداد به جای خودشان در الگو هستند. مثلاً اگر بنویسیم `abc` ، هر رشته‌ای که این سه حرف را پشت سر هم داشته باشد، با الگوی ما مطابقت دارد. اما بعضی علامت‌ها، کاربردهای خاصی در رجکس دارند. برای استفاده از رجکس (RegEx) در پایتون ابتدا باید کتابخانه `re` را به شکل زیر به کدتان اضافه کنید:

```
import re
```

تابع `match`

این تابع به ترتیب دو رشته‌ی `pattern` و `s` را به عنوان ورودی می‌گیرد و بررسی می‌کند که آیا رشته‌ی `s` با رجکس `pattern` انطباق دارد یا خیر. می‌توان از خروجی تابع `match` در یک شرط استفاده کرد. به مثال زیر توجه‌کنید:

```
pattern = '\d\d\d$'
s = input()
if re.match(pattern, s):
    print('matched')
else:
    print('not matched')
```

در این مثال اگر رشته‌ی ورودی یک عدد سه رقمی باشد، `matched` و در غیر این صورت `not matched` چاپ می‌شود.

تابع findall

این تابع دو رشته به عنوان ورودی گرفته و در خروجی یک لیست بر می‌گرداند. این لیست شامل همه زیررشته هایی از رشته دوم است که با الگوی RegEx (رشته اول ورودی) قابل انطباق است. به مثال زیر توجه‌کنید:

```
>>> re.findall("e[a-z]", "regex college")
['eg', 'ex', 'eg']
>>> re.findall("python", "regex college")
[]
```

تابع search

کار این تابع تقریبا شبیه تابع `findall` است با این تفاوت که خروجی‌ای از جنس `Match Object` می‌دهد که به اولین جایی که الگو تکرار شده اشاره می‌کند. تابع‌های مربوط به آن را می‌توانید در مثال زیر مشاهده کنید.

```
>>> x = re.search("e[a-z]", "regex college")
>>> type(x)
<class '_sre.SRE_Match'>
>>> x.string
'regex college'
>>> x.span()
(1, 3)
>>> x.group()
'eg'
```

- تابع `span()` مکان شروع و پایان تکرار را خروجی می‌دهد. (به صورت بسته باز)
- تابع `group()` رشته‌ای که با الگو تطابق پیدا کرده را خروجی می‌دهد.
- اگر هیچ تطابقی پیدا نشود تابع `search` به ما `None` خروجی می‌دهد.

تابع split

عملکرد این تابع مشابه تابع `split` خود پایتون است با این تفاوت که به جای رشته از یک الگو استفاده می‌کند. این تابع یک الگو و یک رشته ورودی می‌گیرد و در مکان‌های تکرار آن الگو رشته را می‌شکند و قسمت‌های به وجود

آمده را در قالب یک لیست بر می‌گرداند. عملیات قسمت کردن از ابتدای رشته شروع میشود و در هر مرحله اگر زیررشته‌ای با شروع از اندیس فعلی وجود داشت که با الگو مطابق باشد آن را حذف و قسمت قبل از الگو را به جواب اضافه می‌کند. اگر چند زیررشته با الگو تطابق داشتند زیررشته‌ای که بیش‌ترین طول را دارد انتخاب می‌شود. همچنین اگر هیچ زیر رشته ای با الگو مطابقت نداشته باشد، یا به تکه انتهایی رشته برسد؛ کل رشته باقی مانده به جواب افزوده می شود.

```
>>> re.split('a', 'salam')
['s', 'l', 'm']
>>> re.split('[0-9]+', 'you92398can820split729with26numbers')
['you', 'can', 'split', 'with', 'numbers']
>>> re.split('[a-z]+', '2093ab20a120dk')
['2093', '20', '120', '']
```

تابع sub

عملکرد این تابع مشابه تابع `replace` پایتون است با این تفاوت که برای پیدا کردن محل‌های تکرار به جای رشته از یک الگو استفاده می‌کند. این تابع یک الگو و دو رشته ورودی می‌گیرد. هر جا که الگو در رشته دوم وجود داشت آن را با رشته اول جایگزین می‌کند. (پیدا کردن این الگو مشابه تابع `split()` است.)

```
>>> re.sub('l', 'salam', 'hello')
'hesalamsalamo'
>>> re.sub('[0-9]', '_digit_', '12ab1')
'_digit__digit_ab_digit_'
>>> re.sub('[0-9]+', '_num_', '12ab1129h')
'_num_ab_num_h'
```

کار با رجکس در پایتون با استفاده از همین ۵ تابع امکان‌پذیر است. چیزی که اهمیت دارد این است که بتوان الگوهای خوبی برای کارکردن با آن پیدا کرد.