# مروری بر عبارات باقاعده

عبارات باقاعده (گاهی به آنها regex نیز می‌گویند) به کاربر اجازه می‌دهد تا با استفاده از هر نوع قانونی که بتوانند تعیین کنند، برای رشته‌ها جستجو کنند. به عنوان مثال، پیدا کردن تمام حروف بزرگ در یک رشته، یا پیدا کردن یک شماره تلفن در یک سند.

عبارات باقاعده به خاطر دستورالعمل‌های غیرقابل پیش‌بینی خود شهرت دارند. این دستورالعمل‌های غیرمعمول نتیجه‌ای از انعطاف‌پذیری آنها هستند. عبارات باقاعده باید قادر باشند هر الگوی رشته‌ای را که می‌توانید تصور کنید، فیلتر کنند، به همین دلیل فرمت پیچیده‌ای دارند.

بیایید با توضیحاتی در مورد جستجوی الگوهای پایه در یک رشته شروع کنیم!

# جستجوی الگوهای پایه

تصور کنید که ما یک رشته به شرح زیر داریم:

In [1]:

```
text = "The person's phone number is 408-555-1234. Call soon!"
```

برای شروع، می‌خواهیم بفهمیم آیا رشته "phone" در داخل رشته متن وجود دارد. می‌توانیم این کار را به سرعت با استفاده از متد زیر انجام دهیم:

In [2]:

```
'phone' in text
```

Out[2]:

```
True
```

اما بیایید قالب عبارات باقاعده را نشان دهیم، زیرا بعداً در جستجوی الگوهایی خواهیم بود که راه حل ساده‌ای ندارند.

In [3]:

```
import re
```

In [5]:

```
pattern = 'phone'
```

In [6]:

```
re.search(pattern, text)
```

Out[6]:

```
<re.Match object; span=(13, 18), match='phone'>
```

In [7]:

```
pattern = "NOT IN TEXT!"
```

In [8]:

```
re.search(pattern, text)
```

همانطور که دیدیم، متد `re.search()` الگو را می‌گیرد، متن را اسکن می‌کند و سپس یک شیء تطبیق را برمی‌گرداند. اگر الگو پیدا نشود، `None` برگردانده می‌شود (در Jupyter Notebook این به معنای این است که هیچ چیزی در زیر سلول نمایش داده نمی‌شود).

بیایید به طور دقیق‌تر به این شیء تطبیق نگاهی بیندازیم.

In [9]:

```
pattern = 'phone'
```

In [10]:

```
match = re.search(pattern, text)
```

In [11]:

```
match
```

Out[11]:

```
<re.Match object; span=(13, 18), match='phone'>
```

اگر دقت کنید می توانید مشاهده نمایید که شی ایجاد شده دارای span می باشد که ایندکس ابتدا و انتهای محلی که الگو یافته شده را برگشته داده است.

In [12]:

```
match.span()
```

Out[12]:

```
(13, 18)
```

In [13]:

```
match.start()
```

Out[13]:

```
13
```

In [14]:

```
1 match.end()
```

Out[14]:

```
18
```

اما در صورتیکه الگو در بیش از یک محل یافت شود، نتیجه چه خواهد بود؟

In [16]:

```
1 text = "my phone is a new phone"
```

In [17]:

```
1 match = re.search("phone", text)
```

In [18]:

```
1 match
```

Out[18]:

```
<re.Match object; span=(3, 8), match='phone'>
```

توجه کنید که فقط بازگویی می‌کند مطابقت را برای اولین وقوع. اگر می‌خواستیم یک لیستی از تمام مطابقت‌ها را دریافت کنیم، می‌توانیم از متد `.findall()` استفاده کنیم:

In [19]:

```
1 matches = re.findall("phone", text)
```

In [20]:

```
1 matches
```

Out[20]:

```
['phone', 'phone']
```

In [21]:

```
1 len(matches)
```

Out[21]:

```
2
```

برای دریافت شیء تطبیق واقعی، از حلقه `for` استفاده کنید:

In [22]:

```
for match in re.finditer("phone", text):
    print(match.span())
```

```
(3, 8)
(18, 23)
```

اگر می‌خواستید متن واقعی که مطابقت داشته را دریافت کنید، می‌توانید از متد `.group()` استفاده کنید.

In [23]:

```
match.group()
```

Out[23]:

```
'phone'
```

# الگوها

تا الان یاد گرفته‌ایم چگونه برای یک رشته ساده جستجو کنیم. اما درباره موارد پیچیده‌تر چه؟ به عنوان مثال جستجوی یک شماره تلفن در یک رشته بزرگ یا یک آدرس ایمیل؟

اگر دقیقاً شماره تلفن یا ایمیل را بدانیم، می‌توانیم از متد جستجو استفاده کنیم. اما اگر ندانیم؟ ممکن است فرمت کلی را بدانیم و می‌توانیم با استفاده از عبارت منظم (Regular Expressions) در مستند برای جستجوی رشته‌هایی که با یک الگوی خاص مطابقت دارند، استفاده کنیم.

در اینجا ممکن است نحوه نوشتار ابتدایی الگوها به ظاهر عجیب و غریب نظر برسد، اما به این موضوع وقت بدهید، اغلب این مسئله تنها مسئله جستجو در کد الگو است.

بیایید شروع کنیم!

## شناسه‌ها برای کاراکترها در الگوها

کاراکترهایی مانند رقم یا یک رشته تکی کدهای مختلفی دارند که آنها را نمایان می‌کنند. می‌توانید از این کدها برای ساختن یک رشته الگو استفاده کنید. توجه کنید که این کدها بسیار از برگشتی `\` استفاده می‌کنند. به همین دلیل زمان تعریف رشته الگو برای عبارت منظم از فرمت زیر استفاده می‌کنیم:

```
r'mypattern'
```

قرار دادن `r` در ابتدای رشته به پایتون می‌گوید که `\` در رشته الگو به عنوان برگشتی فرض نمی‌شود.

در زیر می‌توانید جدولی از تمام شناسه‌های ممکن را مشاهده کنید:

| Character | Description | Example Pattern Code | Exammple Match |
|---|---|---|---|
| \d | A digit | file_\d\d | file_25 |
| \w | Alphanumeric | \w-\w\w\w | A-b_1 |
| \s | White space | a\sb\sc | a b c |

| | | | |
|---|---|---|---|
| ABC | \D\D\D | A non digit | \D |
| (=+-* | \W\W\W\W\W | Non-alphanumeric | \W |
| Yoyo | \S\S\S\S | Non-whitespace | \S |

برای مثال:

In [28]:

```
text = "My telephone number is 408-555-5555"
```

In [29]:

```
phone = re.search(r'\d\d\d-\d\d\d-\d\d\d\d', text)
```

In [30]:

```
phone.group()
```

Out[30]:

```
'408-555-5555'
```

توجه کنید که تکرار \d کمی مزاحم است، به خصوص اگر به دنبال رشته‌های بسیار طولانی از اعداد باشیم. بیایید از ضرایب ممکن استفاده کنیم.

# Quantifiers

حال که ما شناسه‌های ویژه را می‌شناسیم، می‌توانیم از آنها به همراه Quantifiers استفاده کنیم تا تعیین کنیم چه تعدادی از آنها را انتظار داریم.

| Character | Description | Example Pattern Code | Exammple Match |
|---|---|---|---|
| + | Occurs one or more times | Version \w-\w+ | Version A-b1_1 |
| {3} | Occurs exactly 3 times | \D{3} | abc |
| {2,4} | Occurs 2 to 4 times | \d{2,4} | 123 |
| {3,} | Occurs 3 or more | \w{3,} | anycharacters |
| * | Occurs zero or more times | A**B*C | AAACC |
| ? | Once or none | ?plurals | plural |

حالا بیایید الگوی خودمان را بازنویسی کنیم:

In [ ]:

```
phone = re.search(r'\d\d\d-\d\d\d-\d\d\d\d', text)
```

In [31]:

```
re.search(r'\d{3}-\d{3}-\d{4}',text)
```

Out[31]:

```
<re.Match object; span=(23, 35), match='408-555-5555'>
```

In [32]:

```
def multi_re_find(patterns, phrase):
    '''
     Takes in a list of regex patterns
     Prints a list of all matches
     '''
    for pattern in patterns:
        print ('Searching the phrase using the re check: %r' %pattern)
        print (re.findall(pattern,phrase))
        print ('\n')
```

In [40]:

```
test_phrase = 'sdsd..sssddd...sdddsddd...dsds...dssss...sdddd'

test_patterns = [
    'sd*',  # s followed by zero or more d's  s, sd, sdd, sddd, sdddd, ...
    'sd+',  # s followed by one or more d's   sd, sdd, sddd, sdddd, ...
    'sd?',  # s followed by zero or one d's   s, sd
    'sd{3}', # s followed by three d's
    'sd{2,3}', # s followed by two to three d's
]

multi_re_find(test_patterns,test_phrase)
```

```
Searching the phrase using the re check: 'sd*'
['sd', 'sd', 's', 's', 'sddd', 'sddd', 'sddd', 'sd', 's', 's', 's', 's',
's', 's', 'sdddd']


Searching the phrase using the re check: 'sd+'
['sd', 'sd', 'sddd', 'sddd', 'sddd', 'sd', 'sdddd']


Searching the phrase using the re check: 'sd?'
['sd', 'sd', 's', 's', 'sd', 'sd', 'sd', 'sd', 's', 's', 's', 's', 's',
's', 'sd']


Searching the phrase using the re check: 'sd{3}'
['sddd', 'sddd', 'sddd', 'sddd']


Searching the phrase using the re check: 'sd{1,3}'
['sd', 'sd', 'sddd', 'sddd', 'sddd', 'sd', 'sddd']
```

## مجموعه‌های کاراکتری

مجموعه‌های کاراکتری وقتی استفاده می‌شوند که شما می‌خواهید در یک نقطه از ورودی، با هر یک از یک گروه از کاراکترها همخوانی داشته باشید. براکت‌ها برای ساخت ورودی‌های مجموعه‌های کاراکتری استفاده می‌شوند. به عنوان مثال: ورودی [ab] برای جستجوی وقوع هر a یا b استفاده می‌شود. بیایید چند نمونه ببینیم:

In [43]:

```
test_phrase = 'sdsd..sssddd...sdddsddd...dsds...dssss...sdddd'

test_patterns = [
    '[sd]',
    '[sd]+', # sd, sds, sdd, ssd, ds, dss
    '[sd]?'
]

multi_re_find(test_patterns,test_phrase)
```

```
Searching the phrase using the re check: '[sd]'
['s', 'd', 's', 'd', 's', 's', 's', 'd', 'd', 'd', 's', 'd', 'd', 'd',
's', 'd', 'd', 'd', 'd', 's', 'd', 's', 'd', 's', 's', 's', 's', 's', 's',
'd', 'd', 'd', 'd']


Searching the phrase using the re check: '[sd]+'
['sdsd', 'sssddd', 'sdddsddd', 'dsds', 'dssss', 'sdddd']


Searching the phrase using the re check: '[sd]?'
['s', 'd', 's', 'd', '', '', 's', 's', 's', 'd', 'd', 'd', '', '', '',
's', 'd', 'd', 'd', 's', 'd', 'd', 'd', '', '', '', 'd', 's', 'd', 's',
'', '', '', 'd', 's', 's', 's', 's', 's', '', '', '', 's', 'd', 'd', 'd',
'd', '']
```

# گروه‌ها

اگر بخواهیم دو کار انجام دهیم، یعنی پیدا کردن شماره تلفن‌ها و همچنین قادر به استخراج سریع کد منطقه آنها (سه رقم اول) باشیم، می‌توانیم از گروه‌ها برای هر کار عمومی که شامل تجمیع عبارات منظم استفاده کنیم (تا بتوانیم در آینده آنها را تجزیه کنیم).

با استفاده از مثال شماره تلفن، می‌توانیم گروه‌هایی از عبارات منظم را با استفاده از پرانتز تفکیک کنیم:

In [44]:

```
phone_pattern = re.compile(r'(\d{3})-(\d{3})-(\d{4})')
```

In [45]:

```
results = re.search(phone_pattern,text)
```

In [46]:

```
# The entire result
results.group()
```

Out[46]:

```
'408-555-5555'
```

In [47]:

```
# Can then also call by group position.
# remember groups were separated by parenthesis ()
# Something to note is that group ordering starts at 1. Passing in 0 returns everyth
results.group(1)
```

Out[47]:

```
'408'
```

In [48]:

```
results.group(2)
```

Out[48]:

```
'555'
```

In [49]:

```
results.group(3)
```

Out[49]:

```
'5555'
```

In [50]:

```
# We only had three groups of parenthesis
results.group(4)
```

```
---------------------------------------------------------------------------
-
IndexError                                Traceback (most recent call las
t)
Cell In[50], line 2
      1 # We only had three groups of parenthesis
----> 2 results.group(4)

IndexError: no such group
```

# نحوه استفاده از نمادهای دیگر در عبارات منظم

## عملگر OR |

از عملگر پایپ (|) برای داشتن یک عبارت **یا** استفاده کنید. به عنوان مثال:

In [51]:

```
1 re.search(r"man|woman", "This man was here!")
```

Out[51]:

```
<re.Match object; span=(5, 8), match='man'>
```

In [52]:

```
1 re.search(r"man|woman", "This woman was here!")
```

Out[52]:

```
<re.Match object; span=(5, 10), match='woman'>
```

## کاراکتر Wildcard

از "wildcard" به عنوان یک مکانی استفاده کنید که هر کاراکتری را مطابقت خواهد داد. می‌توانید از نقطه ساده (.) برای این منظور استفاده کنید. به عنوان مثال:

In [53]:

```
1 re.findall(r".at", "The cat in the hat sat here")
```

Out[53]:

```
['cat', 'hat', 'sat']
```

In [54]:

```
1 re.findall(r".at","The bat went splat")
```

Out[54]:

```
['bat', 'lat']
```

توجه کنید که ما فقط سه حرف را مطابقت دادیم، این به این دلیل است که برای هر حرف wildcard به یک نقطه (.) نیاز داریم. یا از quantifiers مشروح بالا برای تعیین قوانین خود استفاده کنید.

In [55]:

```
re.findall(r"...at","The bat went splat")
```

Out[55]:

```
['e bat', 'splat']
```

اما این همچنان مشکل بازگرفتن بیشتر را با خود به همراه می‌آورد. در واقع، ما فقط می‌خواهیم کلماتی را که با "at" ختم می‌شوند، دریافت کنیم.

In [56]:

```
# One or more non-whitespace that ends with 'at'
re.findall(r"\S+at", "The bat went splat")
```

Out[56]:

```
['bat', 'splat']
```

## شروع با و پایان با

ما می‌توانیم از علامت ^ برای نشان دادن شروع با استفاده کنیم، و از علامت $ برای نشان دادن پایان با:

In [63]:

```
# Ends with a number
re.findall(r'\d$','This 32 ends with a number 2')
```

Out[63]:

```
['52']
```

In [61]:

```
# Starts with a number
re.findall(r'^\d','1 is the loneliest number.')
```

Out[61]:

```
['1']
```

توجه کنید که این برای تمام رشته است، نه کلمات فردی!

## استثنا

برای استثنای کاراکترها، می‌توانیم از علامت ^ در ترکیب با یک مجموعه از براکت‌ها [] استفاده کنیم. هر چیزی درون براکت‌ها استثنا می‌شود. به عنوان مثال:

In [65]:

```
1 phrase = "there are 3 numbers 34 inside 5 this sentence."
```

In [66]:

```
1 re.findall(r'[^\d]',phrase)
```

Out[66]:

```
['t',
 'h',
 'e',
 'r',
 'e',
 ' ',
 'a',
 'r',
 'e',
 ' ',
 ' ',
 ' ',
 'n',
 'u',
 'm',
 'b',
 'e',
 'r',
 's',
 ' ',
 ' ',
 ' ',
 'i',
 'n',
 's',
 'i',
 'd',
 'e',
 ' ',
 ' ',
 ' ',
 't',
 'h',
 'i',
 's',
 ' ',
 ' ',
 's',
 'e',
 'n',
 't',
 'e',
 'n',
 'c',
 'e',
 '.']
```

برای دریافت خروجی در قالب کلمات از علامت + استفاده می کنیم:

In [67]:

```
1 re.findall(r'[^\d]+',phrase)
```

Out[67]:

```
['there are ', ' numbers ', ' inside ', ' this sentence.']
```

از حالت می توانیم برای حذف punctuation ها در متون استفاده کنیم

In [68]:

```
1 test_phrase = 'This is a string! But it has punctuation. How can we remove it?'
```

In [69]:

```
1 re.findall('[^!.? ]+',test_phrase)
```

Out[69]:

```
['This',
 'is',
 'a',
 'string',
 'But',
 'it',
 'has',
 'punctuation',
 'How',
 'can',
 'we',
 'remove',
 'it']
```

In [70]:

```
1 clean = ' '.join(re.findall('[^!.? ]+',test_phrase))
```

In [71]:

```
1 clean
```

Out[71]:

```
'This is a string But it has punctuation How can we remove it'
```

## پرانتز برای چندین گزینه

اگر برای تطبیق با چندین گزینه، می‌توانیم از پرانتز استفاده کنیم تا این گزینه‌ها را لیست کنیم. به عنوان مثال:

In [72]:

```
# Find words that start with cat and end with one of these options: 'fish','nap', or
text = 'Hello, would you like some catfish?'
texttwo = "Hello, would you like to take a catnap?"
textthree = "Hello, have you seen this caterpillar?"
```

In [73]:

```
re.search(r'cat(fish|nap|claw)',text)
```

Out[73]:

```
<re.Match object; span=(27, 34), match='catfish'>
```

In [74]:

```
re.search(r'cat(fish|nap|claw)',texttwo)
```

Out[74]:

```
<re.Match object; span=(32, 38), match='catnap'>
```

In [75]:

```
# None returned
re.search(r'cat(fish|nap|claw)',textthree)
```

## محدوده کاراکترها

همانطور که مجموعه‌های کاراکتری بزرگتر می‌شوند، تایپ کردن هر کاراکتری که باید (یا نباید) متناسب باشد، خیلی خسته کننده می‌شود. با استفاده از فرمت محدوده کاراکترها می‌توانید مجموعه کاراکتری را تعریف کنید که همهٔ کاراکترهای متوالی بین یک نقطه شروع و پایان را شامل شود. فرمت استفاده شده برای این منظور [شروع-پایان] است. مورد استفاده رایج برای جستجوی یک محدودهٔ خاص از حروف الفباست، مانند [a-f] که هر نمونه حروفی بین a و f را برمی‌گرداند.

بیایید به سراغ برخی از مثال‌ها برویم:

In [76]:

```
test_phrase = 'This is an example sentence. Lets see if we can find some letters.'
test_patterns=[ '[a-z]+', # sequences of lower case letters
                '[A-Z]+', # sequences of upper case letters
                '[a-zA-Z]+', # sequences of lower or upper case letters
                '[A-Z][a-z]+'] # one upper case letter followed by lower case lette

multi_re_find(test_patterns,test_phrase)
```

```
Searching the phrase using the re check: '[a-z]+'
['his', 'is', 'an', 'example', 'sentence', 'ets', 'see', 'if', 'we', 'ca
n', 'find', 'some', 'letters']


Searching the phrase using the re check: '[A-Z]+'
['T', 'L']


Searching the phrase using the re check: '[a-zA-Z]+'
['This', 'is', 'an', 'example', 'sentence', 'Lets', 'see', 'if', 'we', 'ca
n', 'find', 'some', 'letters']


Searching the phrase using the re check: '[A-Z][a-z]+'
['This', 'Lets']
```

## جمع بندی

کارتان عالی بود! برای اطلاعات کامل در مورد همهٔ الگوهای ممکن، به آدرس زیر مراجعه کنید:

https://docs.python.org/3/howto/regex.html (https://docs.python.org/3/howto/regex.html)