Cheatography

python regex(regular expression) Cheat Sheet

by Nima (nimakarimian) via cheatography.com/113429/cs/23788/

re.match()

```
re.match(pattern,
"spamspamspam")
#returns True
```

match returns an object representing the match, if not, it returns None.

Sub()

re.sub(pattern, repl, string, count=0)

```
str = "My name is David. Hi
David."
pattern = r"Da vid "
newstr = re.sub (pa ttern, " -
Amy ", str)
print( newstr)
>>>
My name is Amy. Hi Amy.
>>>
```

This method replaces all occurrences of the pattern in string with repl, substituting all occurrences, unless count provided. This method returns the modified string.

^start &end

```
pattern = r"^gr.y$"
```

The next two metacharacters are ^ and $. These match the start and end of a string, respectively.

[] character classes 3

```
pattern = r"[^ A-Z ]"
if re.sea rch (pa ttern, "this
is all quiet"):
      pri nt( " Match 1")
if re.sea rch (pa ttern, " AbC -
dEf G12 3"):
      pri nt( " Match 2")
if re.sea rch (pa ttern, " THI -
SIS ALL SHO UTI NG"):
      pri nt( " Match 3")
##The pattern [^A-Z] excludes
uppercase strings.
Note, that the ^ should be
inside the brackets to invert
the character class.
>>>
Match 1
Match 2
>>>
```

special sequences

```
pattern = r"(.+) \1"
match = re.mat ch( pat tern,
"word word")
if match:
      print ("Match 1")
match = re.mat ch( pat tern, "?!
?!")
if match:
      print ("Match 2")
match = re.mat ch( pat tern,
"abc cde")
if match:
      print ("Match 3")
>>>
Match 1
```

special sequences (cont)

```
> Match 2
>>>
```

Note, that "(.+) \1" is not the same as "(.+) (.+)", because \1 refers to the first group's subexpression, which is the matched expression itself, and not the regex pattern.

There are various special sequences you can use in regular expressions. They are written as a backslash followed by another character.

One useful special sequence is a backslash and a number between 1 and 99, e.g., \1 or \17. This matches the expression of the group of that number.

? metacharacter

```
pattern = r"ic e(- )?c rea m"
if re.mat ch( pat tern, " ice -
cr eam "):
      pri nt( " Match 1")
if re.mat ch( pat tern, " ice -
cre am"):
      pri nt( " Match 2")
```

The metacharacter ? means "zero or one repetitions".

{} metacharacters

```
pattern = r"9{1,3}$"
if re.mat ch( pat tern, " 9"):
      pri nt( " Match 1")
if re.mat ch( pat tern, " -
999 "):
      pri nt( " Match 2")
```

By **Nima** (nimakarimian)

cheatography.com/nimakarimian/
www.nimakarimian.ir

Published 21st July, 2020.
Last updated 21st July, 2020.

Sponsored by **ApolloPad.com**
Everyone has a novel in them. Finish Yours!
https://apollopad.com

{} metacharacters (cont)

```
> if re.match(pattern, "9999"):
  print("Match 3")
>>>
Match 1
Match 2
>>>
```

Curly braces can be used to represent the number of repetitions between two numbers.
The regex {x,y} means "between x and y repetitions of something".
Hence {0,1} is the same thing as ?.
If the first number is missing, it is taken to be zero. If the second number is missing, it is taken to be infinity.

search() and findall()

```
if re.search(pattern,
"eggspamsausagespam"):
      pri nt( " Mat ch")
else:
      pri nt( "No match")

print( re.f in dal l(p attern,
" egg spa msa usa ges pam "))
>>>
Match
['spam', 'spam']
>>>
```

The function re.search finds a match of a pattern anywhere in the string.
The function re.findall returns a list of all substrings that match a pattern.

. (dot).

```
pattern = r"gr.y"
# this will be grey or gray or
anything else except newline
```

This matches any character, other than a new line.

\d \s \w Special sequences

```
pattern = r"(\D+\d)"
match = re.mat ch( pat tern, "Hi
999!")
if match:
      pri nt( " Match 1")
match = re.mat ch( pat tern, "1,
23, 456!")
if match:
      pri nt( " Match 2")
match = re.mat ch( pat tern, " !
$?")
if match:
       pri nt( " Match 3")
>>>
Match 1
>>>
```

More useful special sequences are \d, \s, and \w.
These match digits, whitespace, and word characters respectively.
In ASCII mode they are equivalent to [0-9], [\t\n\r\f\v], and [a-zA-Z0-9_].
In Unicode mode they match certain other characters, as well. For instance, \w matches letters with accents.
Versions of these special sequences with upper case letters - \D, \S, and \W - mean the opposite to the lower-case versions. For instance, \D matches anything that isn't a digit.

[] character classes 2

```
pattern = r"[A -Z] [A- Z][ 0-
9 ]"
if re.sea rch (pa ttern, " -
LS8 "):
      pri nt( " Match 1")
if re.sea rch (pa ttern, " E3"):
      pri nt( " Match 2")
#The pattern in the example
above matches strings that
contain two alphabetic uppercase
letters followed by a digit.
>>>
Match 1
>>>
```

Character classes can also match ranges of characters.
The class [a-z] matches any lowercase alphabetic character.
The class [G-P] matches any uppercase character from G to P.
The class [0-9] matches any digit.
Multiple ranges can be included in one class. For example, [A-Za-z] matches a letter of any cases.

+ metacharacter

```
pattern = r"g+ "
if re.mat ch( pat tern, " g"):
      pri nt( " Match 1")
To summarize:
* matches 0 or more occurr ences
of the preceding expres sion.
+ matches 1 or more occurrence
of the preceding expres sion.
```

The metacharacter + is very similar to *, except it means "one or more repetitions", as opposed to "zero or more repetitions".

By **Nima** (nimakarimian)

cheatography.com/nimakarimian/
www.nimakarimian.ir

Published 21st July, 2020.
Last updated 21st July, 2020.

Sponsored by **ApolloPad.com**
Everyone has a novel in them. Finish Yours!
https://apollopad.com

Groups in metacharacters ()

```
pattern = r"a(bc)(de)(f(g)h)i"
match = re.mat ch( pat tern, " -
abc def ghi jkl mno p")
if match:
      pri nt( mat ch.g ro up())
      pri nt( mat ch.g ro up(0))
      pri nt( mat ch.g ro up(1))
      pri nt( mat ch.g ro up(2))
      pri nt( mat ch.g ro ups())
>>>
abcdefghi
abcdefghi
bc
de
('bc', 'de', 'fgh', 'g')
>>>
```

The content of groups in a match can be accessed using the group function.
A call of group(0) or group() returns the whole match.
A call of group(n), where n is greater than 0, returns the nth group from the left.
The method groups() returns all groups up from 1.

\A \Z \b special sequences

```
pattern = r"\b (ca t) \b"
match = re.sea rch (pa ttern,
"The cat sat!")
if match:
      print ("Match 1")
match = re.sea rch (pa ttern,
"We s>c at< ter ed? ")
```

\A \Z \b special sequences (cont)

```
> if match:
   print ("Match 2")
match = re.search(pattern, "We scattered.")
if match:
   print ("Match 3")
>>>
Match 1
Match 2
>>>
```

"\b(cat)\b" basically matches the word "cat" surrounded by word boundaries.

Additional special sequences are \A, \Z, and \b.
The sequences \A and \Z match the beginning and end of a string, respectively.
The sequence \b matches the empty string between \w and \W characters, or \w characters and the beginning or end of the string. Informally, it represents the boundary between words.
The sequence \B matches the empty string anywhere else.

| "or" metacharacter

```
pattern = r"gr (a| e)y "
match = re.mat ch( pat tern, " -
gra y")
if match:
      print ("Match 1")
match = re.mat ch( pat tern, " -
gre y")
if match:
      print ("Match 2")
```

| "or" metacharacter (cont)

```
> match = re.match(pattern, "griy")
if match:
   print ("Match 3")
>>>
Match 1
Match 2
>>>
```

Another important metacharacter is |.
This means "or", so red|blue matches either "red" or "blue".

named groups & noncapturing groups

```
pattern = r"(?P<first>abc)
(?:def)(ghi)"
match = re.mat ch( pat tern, " -
abc def ghi ")
if match:
      pri nt( mat ch.g ro -
up( " fir st"))
      pri nt( mat ch.g ro ups())
>>>
abc
('abc', 'ghi')
>>>
```

Named groups have the format (?P<name>...), where name is the name of the group, and ... is the content. They behave exactly the same as normal groups, except they can be accessed by group(name) in addition to its number.
Non-capturing groups have the format (?:...). They are not accessible by the group method, so they can be added to an existing regular expression without breaking the numbering.

By **Nima** (nimakarimian)

cheatography.com/nimakarimian/
www.nimakarimian.ir

Published 21st July, 2020.
Last updated 21st July, 2020.

Sponsored by **ApolloPad.com**
Everyone has a novel in them. Finish Yours!
https://apollopad.com

Cheatography

python regex(regular expression) Cheat Sheet

by Nima (nimakarimian) via cheatography.com/113429/cs/23788/

* metacharacter

```
pattern = r"eg g(s pam )*"
if re.mat ch( pat tern, " egg "):
      pri nt( " Match 1")
if re.mat ch( pat tern, " egg spa mspam spa m"):
      pri nt( " Match 2")
if re.mat ch( pat tern, " spa m"):
      pri nt( " Match 3")
>>>
match 1
match 2
>>>
The example above matches strings that start
with " egg " and follow with zero or more "
spa m"s.
.....
```

The metacharacter * means "zero or more repetitions of the previous thing".

[] character classes

```
pattern = r"[aeiou]"
if re.sea rch (pa ttern, " gre -
y"):
      pri nt( " Match 1")
if re.sea rch (pa ttern, " qwe -
rty uio p"):
      pri nt( " Match 2")
if re.sea rch (pa ttern, " -
rhythm myths"):
      pri nt( " Match 3")
#The pattern [aeiou] in the
search function matches all
strings that contain any one of
the characters defined
```

[] character classes (cont)

```
> >>>
Match 1
Match 2
>>>
```

Character classes provide a way to match only one of a specific set of characters.

Search->>Group, Start,End,Span

```
match = re.search(pattern,
"eggspamsausage")
if match:
      pri nt( mat ch.g ro up())
      pri nt( mat ch.s ta rt())
      pri nt( mat ch.e nd())
      pri nt( mat ch.s pan())
>>>.....
pam
4
7
(4, 7)
>>>
```

The regex search returns an object with several methods that give details about it. These methods include group which returns the string matched, start and end which return the start and ending positions of the first match, and span which returns the start and end positions of the first match as a tuple.

By **Nima** (nimakarimian)

cheatography.com/nimakarimian/

www.nimakarimian.ir

Published 21st July, 2020.

Last updated 21st July, 2020.

Sponsored by **ApolloPad.com**

Everyone has a novel in them. Finish Yours!

https://apollopad.com