regular-expressions

February 12, 2021

1 Regular Expressions in Python

Author: samsar4

1.0.1 Brief Introduction

Regular expressions (called REs, or regexes, or regex patterns) are essentially a tiny, highly specialized programming language embedded inside Python and made available through the re module. Using this little language, you specify the rules for the set of possible strings that you want to match; this set might contain English sentences, or e-mail addresses, or TeX commands, or anything you like. You can then ask questions such as "Does this string match the pattern?", or "Is there a match for the pattern anywhere in this string?". You can also use REs to modify a string or to split it apart in various ways.

1.0.2 Special characters:

```
\   escape special characters
.   matches any character
^   matches beginning of string
$   matches end of string
[5b-d]  matches any chars '5', 'b', 'c' or 'd'
[^a-c6] matches any char except 'a', 'b', 'c' or '6'
R|S matches either regex R or regex S
()  creates a capture group and indicates precedence
```

1.0.3 Special sequences:

```
\A  start of string
\b  matches empty string at word boundary (between \w and \W)
\B  matches empty string not at word boundary
\d  digit
\D  non-digit
\s  whitespace: [ \t\n\r\f\v]
\S  non-whitespace
\w  alphanumeric: [0-9a-zA-Z_]
\W  non-alphanumeric
\Z  end of string
\g<id>  matches a previously defined group
```

1.0.4 Quantifiers

```
*   0 or more (append ? for non-greedy)
+   1 or more (append ? for non-greedy)
?   0 or 1 (append ? for non-greedy)
{m} exactly mm occurrences
{m, n}  from m to n. m defaults to 0, n to infinity
{m, n}? from m to n, as few as possible
```

[1]:
```python
import re
```

1.0.5 Grabbing the IP addresses with `findall()` method

`findall()` method: The string is scanned left-to-right, and matches are returned in the order found. If one or more groups are present in the pattern, return a list of groups; this will be a list of tuples if the pattern has more than one group. Empty matches are included in the result.

[69]:
```python
log = 'Oct 31 06:11:35 gw1 kernel: Â OUTBOUND IN=eth1 OUT=eth0 SRC=192.168.1.
↪101 DST=204.152.189.116 LEN=52 TOS=0x00 PREC=0x00 TTL=127 ID=18437 DF␣
↪PROTO=TCP SPT=32865 DPT=80 WINDOW=5840 RES=0x00 SYN URGP=0'
print(log)
```

```
Oct 31 06:11:35 gw1 kernel: Â OUTBOUND IN=eth1 OUT=eth0 SRC=192.168.1.101
DST=204.152.189.116 LEN=52 TOS=0x00 PREC=0x00 TTL=127 ID=18437 DF PROTO=TCP
SPT=32865 DPT=80 WINDOW=5840 RES=0x00 SYN URGP=0
```

[91]:
```python
ip_source, ip_dest = re.findall(r"\b(?:\d{1,3}\.){3}\d{1,3}\b",log)

print(f"The Source IP address is: {ip_source}\nThe Destination IP address is:␣
↪{ip_dest}")
```

```
The Source IP address is: 192.168.1.101
The Destination IP address is: 204.152.189.116
```

1.0.6 Using `findall()` to match (!) and (.) chars

[8]:
```python
sentence = "This is a sentence that ends with a period!!! ..."
result = re.findall(r"[^a-zA-Z ]", sentence)
print(result)
```

```
['!', '!', '!', '.', '.', '.']
```

`search()` method: Scan through a string, looking for any location where this RE matches; Returns a single value that first encounters.

- Match by word, two words, but one result:

[9]:
```python
print(re.search(r"that|ends", sentence))
```

```
<re.Match object; span=(19, 23), match='that'>
```

- To output more than one value, use `findall()`:

```
[43]: print(re.findall(r"that|ends", sentence))
```

```
['that', 'ends']
```

- Expanding search (greedy)

```
[84]: string = "Python Programming"


print(re.search(r"Py.*ng", string))
```

```
<re.Match object; span=(0, 18), match='Python Programming'>
```

- Restricting search for non-greedy

```
[46]: print(re.search(r"Py[a-z]*n", string))
```

```
<re.Match object; span=(0, 6), match='Python'>
```

- Append for non-greedy

```
[48]: print(re.search(r"t+h+o+n", string))
```

```
<re.Match object; span=(2, 6), match='thon'>
```

```
[54]: strings_a = "Animal Kingdom"


print(re.search(r"[Aa].*[Aa]", strings_a))
```

```
<re.Match object; span=(0, 5), match='Anima'>
```

- Matching the condition

```
[60]: strings_b = "Animal Kingdom is Amazing!"

print(re.search(r"A?mazing", strings_b))
```

```
<re.Match object; span=(18, 25), match='Amazing'>
```

- Checking vowels if they are between other characters:

1.1 Examples

1.1.1 Matching acronym:

```
[92]: # Find acronym and return True if matches
def contains_acronym(text):
  pattern = "[()]"
  result = re.search(pattern, text)
  return result != None

print(contains_acronym("Instant messaging (IM) is a set of communication␣
↪technologies used for text-based communication"))
print(contains_acronym("American Standard Code for Information Interchange␣
↪(ASCII) is a character encoding standard for electronic communication"))
print(contains_acronym("Please do NOT enter without permission!"))
print(contains_acronym("A man cannot be comfortable without his own approval."))
print(contains_acronym("PostScript is a fourth-generation programming language␣
↪(4GL)"))
print(contains_acronym("Have fun using a self-contained underwater breathing␣
↪apparatus (Scuba)!"))
```

```
True
True
False
False
True
True
```

1.1.2 Combining multiple matching parameters:

1. Starts with an uppercase letter
2. At least some lowercase letters or a space
3. Ends with a period, question mark, or exclamation point

[93]:
```
def check_sentence(text):
  result_3 = re.search(r"^[A-Z][a-z! ]*[.?!]$", text)
  return result_3 != None

print(check_sentence("Is this is a sentence?"))
print(check_sentence("is this is a sentence?"))
print(check_sentence("Hello"))
print(check_sentence("1-2-3-GO!"))
print(check_sentence("A star is born."))
```

```
True
False
False
False
True
```

1. Contains alphanumeric characters (which includes letters, numbers, and underscores)
2. Periods, dashes, and a plus sign
3. Character-only top-level domain such as ".com", ".info", ".edu", etc.

[95]:
```
def check_web_address(text):
  pattern = "[\w\._-]*\.[A-Za-z]*$"
  result_4 = re.search(pattern, text)
  return result_4 != None

print(check_web_address("gmail.com"))
print(check_web_address("www@google"))
print(check_web_address("www.mrrobot.com"))
print(check_web_address("web-address.com/homepage"))
print(check_web_address("Some_random_blog.US"))
```

```
True
False
True
False
True
```

- Matching the time format of a 12-hour clock

[99]:
```
import re
def check_time(text):
  pattern = '^(1[0-2]|1?[1-9]):([0-5][0-9])( ?([AaPp][Mm]))'
  result = re.search(pattern, text)
  return result != None

print(check_time("12:45pm")) # True
print(check_time("9:59 AM")) # True
print(check_time("6:60am")) # False
print(check_time("five o'clock")) # False
```

```
True
True
False
False
```

1.1.3 Grabbing a single email address:

[83]:
```
text = "Feb 23 12:54:06, Received an email from johnwick_1964@gmail.com"

## search() method: for scanning through the string
print(re.search(r"[A-Za-z0-9-_]+@[A-Za-z0-9].+[a-z]", text))

## findall() method: to return the match into a list
print(re.findall(r"[A-Za-z0-9-_]+@[A-Za-z0-9].+[a-z]", text))
```

```
<re.Match object; span=(40, 63), match='johnwick_1964@gmail.com'>
['johnwick_1964@gmail.com']
```

1.1.4 Capturing groups:

Portions of the pattern that are enclosed in parentheses

[101]:
```
name = "John, Wick"

# scanning
result_6 = re.search(r"^(\w*), (\w*)$", name)

# using groups() method returns a tuple
print(result_6.groups())
```

```
('John', 'Wick')
```

[100]:
```
# Acessing indexes from the tuple
print(f"Hello, {result_6[1]} {result_6[2]}")
```

```
Hello, John Wick
```

1.1.5 Extracting a PID from a syslog line

[120]:
```
log_2 = "May 3 05:55:45 mycomputer bad_process[12345]: ERROR Performing package␣
 ↪upgrade"
regex_2 = r"\[(\d+)\]"
result_7 = re.search(regex_2, log_2)
print(result_7[1])
```

```
12345
```

To avoid a simple error in case of some value have different characters instead of numbers inside the square brackets `[]`

[145]:
```
def extract_pid(log_line):
    regex_2 = r"\[(\d+)\]"
    result_7 = re.search(regex_2, log_line)
    if result_7 is None:
        return ""
    return result_7[1]
```

[146]:
```
print(extract_pid(log_2))
```

```
12345
```

[149]:
```
print(extract_pid("13 Elephant's in a [cage]"))
```

1.1.6 Extracting Date, Time and PID from syslog lines

[5]:
```
def show_time_of_pid(line):
  pattern = r"([a-zA-Z]+ \d+ \d+:\d+:\d+).*\[(\d+)\]\:"
  result = re.search(pattern, line)
  return f"{result.group(1)} PID: {result.group(2)}"

print(show_time_of_pid("Jul 6 14:01:23 computer.name CRON[29440]: USER␣
 ↪(good_user)"))
print(show_time_of_pid("Jul 6 14:02:08 computer.name jam_tag=psim[29187]: (UUID:
 ↪006)"))
print(show_time_of_pid("Jul 6 14:02:09 computer.name jam_tag=psim[29187]: (UUID:
 ↪007)"))
print(show_time_of_pid("Jul 6 14:03:01 computer.name CRON[29440]: USER␣
 ↪(naughty_user)"))
print(show_time_of_pid("Jul 6 14:03:40 computer.name cacheclient[29807]: start␣
 ↪syncing from \"0xDEADBEEF\""))
print(show_time_of_pid("Jul 6 14:04:01 computer.name CRON[29440]: USER␣
 ↪(naughty_user)"))
print(show_time_of_pid("Jul 6 14:05:01 computer.name CRON[29440]: USER␣
 ↪(naughty_user)"))
```

```
Jul 6 14:01:23 PID: 29440
Jul 6 14:02:08 PID: 29187
Jul 6 14:02:09 PID: 29187
Jul 6 14:03:01 PID: 29440
Jul 6 14:03:40 PID: 29807
Jul 6 14:04:01 PID: 29440
Jul 6 14:05:01 PID: 29440
```

1.1.7 Splitting and Replacing

split() method: the string into a list, splitting it wherever the RE matches

sub() method: Find all substrings where the RE matches, and replace them with a different string

[153]:
```
re.split(r"[.?!]", "May 3 05:55:45, mycomputer, bad_process[12345]:, ERROR␣
 ↪Performing package upgrade")
```

[153]:
```
['May 3 05:55:45, mycomputer, bad_process[12345]:, ERROR Performing package
upgrade']
```

Adding commans and notation marks as elements of a list:

[156]:
```
re.split(r"([.?!])", "May 3 05:55:45! mycomputer, bad_process[12345]:, ERROR␣
 ↪Performing package upgrade?")
```

[156]:
```
['May 3 05:55:45',
 '!',
 ' mycomputer, bad_process[12345]:, ERROR Performing package upgrade',
 '?',
 '']
```

1.1.8 Replacing

Example: **Hiding an e-mail address from log files**

```
[69]: ip_quote = "Your IP address 192.168.1.12 is a private address class C"
      print(re.search(r"[\d0-9]+.[\d0-9]+.[\d0-9]+.[\d0-9].", ip_quote))
      print(re.findall(r"[\d0-9]+.[\d0-9]+.[\d0-9]+.[\d0-9].", ip_quote))
```

```
<re.Match object; span=(16, 28), match='192.168.1.12'>
['192.168.1.12']
```

```
[41]: re.sub(r"[\w.%+-]+@[\w.-]+", "[REDACTED]", "Received an email for␣
       ↪johnwick_1964@gmail.com")
```

```
[41]: 'Received an email for [REDACTED]'
```

1.1.9 References & Useful links

- https://pythex.org/
- https://docs.python.org/3/howto/regex.html
- https://regex101.com/
- https://docs.python.org/3/howto/regex.html
- https://docs.python.org/3/library/re.html
- https://docs.python.org/3/howto/regex.html#greedy-versus-non-greedy
- https://regexcrossword.com/