3.4 REGULAR EXPRESSIONS

Searching for required patterns and extracting only the lines/words matching the pattern is a very common task in solving problems programmatically. We have done such tasks earlier using string slicing and string methods like *split()*, *find()* etc. As the task of searching and extracting is very common, Python provides a powerful library called ***regular expressions*** to handle these tasks elegantly. Though they have quite complicated syntax, they provide efficient way of searching the patterns.

The regular expressions are themselves little programs to search and parse strings. To use them in our program, the library/module ***re*** must be imported. There is a ***search()*** function in this module, which is used to find particular substring within a string. Consider the following example –

```
import re
fhand = open('myfile.txt')
for line in fhand:
    line = line.rstrip()
    if re.search('how', line):
        print(line)
```

By referring to file *myfile.txt* that has been discussed in previous Chapters, the output would be –

```
hello, how are you?
how about you?
```

In the above program, the *search()* function is used to search the lines containing a word *how*.

One can observe that the above program is not much different from a program that uses ***find()*** function of strings. But, regular expressions make use of special characters with specific meaning. In the following example, we make use of caret (^) symbol, which indicates beginning of the line.

```
import re
hand = open('myfile.txt')
for line in hand:
    line = line.rstrip()
    if re.search('^how', line):
        print(line)
```

The output would be –

```
how about you?
```

Here, we have searched for a line which starts with a string *how.* Again, this program will not makes use of regular expression fully. Because, the above program would have been

written using a string function *startswith()*. Hence, in the next section, we will understand the true usage of regular expressions.

3.4.1 Character Matching in Regular Expressions

Python provides a list of meta-characters to match search strings. Table 3.1 shows the details of few important metacharacters. Some of the examples for quick and easy understanding of regular expressions are given in Table 3.2.

Table 3.1 List of Important Meta-Characters

Character	Meaning
^ (caret)	Matches beginning of the line
$	Matches end of the line
. (dot)	Matches any single character except newline. Using option *m*, then newline also can be matched
[…]	Matches any single character in brackets
[^…]	Matches any single character NOT in brackets
re*	Matches 0 or more occurrences of preceding expression.
re+	Matches 1 or more occurrence of preceding expression.
re?	Matches 0 or 1 occurrence of preceding expression.
re{ n}	Matches exactly n number of occurrences of preceding expression.
re{ n,}	Matches n or more occurrences of preceding expression.
re{ n, m}	Matches at least n and at most m occurrences of preceding expression.
a| b	Matches either a or b.
(re)	Groups regular expressions and remembers matched text.
\d	Matches digits. Equivalent to [0-9].
\D	Matches non-digits.
\w	Matches word characters.
\W	Matches non-word characters.
\s	Matches whitespace. Equivalent to [\t\n\r\f].
\S	Matches non-whitespace.
\A	Matches beginning of string.
\Z	Matches end of string. If a newline exists, it matches just before newline.
\z	Matches end of string.
\b	Matches the empty string, but only at the start or end of a word.
\B	Matches the empty string, but not at the start or end of a word.
( )	When parentheses are added to a regular expression, they are ignored for the purpose of matching, but allow you to extract a particular subset of the matched string rather than the whole string when using findall()

Table 3.2 Examples for Regular Expressions

Expression	Description
`[Pp]ython`	Match "Python" or "python"
`rub[ye]`	Match "ruby" or "rube"
`[aeiou]`	Match any one lowercase vowel
`[0-9]`	Match any digit; same as [0123456789]
`[a-z]`	Match any lowercase ASCII letter
`[A-Z]`	Match any uppercase ASCII letter
`[a-zA-Z0-9]`	Match any of uppercase, lowercase alphabets and digits
`[^aeiou]`	Match anything other than a lowercase vowel
`[^0-9]`	Match anything other than a digit

Most commonly used metacharacter is dot, which matches any character. Consider the following example, where the regular expression is for searching lines which starts with `I` and has any two characters (any character represented by two dots) and then has a character `m`.

```
import re
fhand = open('myfile.txt')
for line in fhand:
    line = line.rstrip()
    if re.search('^I..m', line):
          print(line)
```

The output would be –

```
I am doing fine.
```

Note that, the regular expression `^I..m` not only matches '`I am`', but it can match '`Isdm`', '`I*3m`' and so on. That is, between `I` and `m`, there can be any two characters.

In the previous program, we knew that there are exactly two characters between I and m. Hence, we could able to give two dots. But, when we don't know the exact number of characters between two characters (or strings), we can make use of dot and + symbols together. Consider the below given program –

```
import re
hand = open('myfile.txt')
for line in hand:
    line = line.rstrip()
    if re.search('^h.+u', line):
        print(line)
```

The output would be –

```
hello, how are you?
how about you?
```

Observe the regular expression `^h.+u` here. It indicates that, the string should be starting with h and ending with u and there may by any number of (dot and +) characters in-between.

Few examples:
To understand the behavior of few basic meta characters, we will see some examples. The file used for these examples is *mbox-short.txt* which can be downloaded from –
https://www.py4e.com/code3/mbox-short.txt

Use this as input and try following examples –

- **Pattern to extract lines starting with the word *From* (or *from*) and ending with *edu*:**

```
import re
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    pattern = '^[Ff]rom.*edu$'
    if re.search(pattern, line):
        print(line)
```

  Here the pattern given for regular expression indicates that the line should start with either *From* or *from*. Then there may be 0 or more characters, and later the line should end with *edu.*

- **Pattern to extract lines ending with any digit:**
  Replace the `pattern` by following string, rest of the program will remain the same.

```
pattern = '[0-9]$'
```

- **Using *Not* :**

```
pattern = '^[^a-z0-9]+'
```

  Here, the first ^ indicates we want something to match in the beginning of a line. Then, the ^ inside square-brackets indicate *do not match any single character within bracket*. Hence, the whole meaning would be – line must be started with anything other than a lower-case alphabets and digits. In other words, the line should not be started with lowercase alphabet and digits.

- **Start with upper case letters and end with digits:**

```
pattern = '^[A-Z].*[0-9]$'
```

  Here, the line should start with capital letters, followed by 0 or more characters, but must end with any digit.

3.4.2 Extracting Data using Regular Expressions

Python provides a method ***findall()*** to extract all of the substrings matching a regular expression. This function returns a list of all non-overlapping matches in the string. If there is no match found, the function returns an empty list. Consider an example of extracting anything that looks like an email address from any line.

```
import re
s = 'A message from csev@umich.edu to cwen@iupui.edu about meeting
     @2PM'
lst = re.findall('\S+@\S+', s)
print(lst)
```

The output would be –

```
['csev@umich.edu', 'cwen@iupui.edu']
```

Here, the pattern indicates at least one non-white space characters (`\S`) before @ and at least one non-white space after @. Hence, it will not match with `@2pm`, because of a white-space before @.

Now, we can write a complete program to extract all email-ids from the file.

```
import re
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    x = re.findall('\S+@\S+', line)
    if len(x) > 0:
        print(x)
```

Here, the condition `len(x) > 0` is checked because, we want to print only the line which contain an email-ID. If any line do not find the match for a pattern given, the *findall()* function will return an empty list. The length of empty list will be zero, and hence we would like to print the lines only with length greater than 0.

The output of above program will be something as below –

```
['stephen.marquard@uct.ac.za']
['<postmaster@collab.sakaiproject.org>']
['<200801051412.m05ECIaH010327@nakamura.uits.iupui.edu>']
['<source@collab.sakaiproject.org>;']
['<source@collab.sakaiproject.org>;']
['<source@collab.sakaiproject.org>;']
['apache@localhost)']
.....
.....
```

Note that, apart from just email-ID's, the output contains additional characters (<, >, ; etc) attached to the extracted pattern. To remove all that, refine the pattern. That is, we want email-ID to be started with any alphabets or digits, and ending with only alphabets. Hence, the statement would be –

```
x = re.findall('[a-zA-Z0-9]\S*@\S*[a-zA-Z]', line)
```

3.4.3 Combining Searching and Extracting

Assume that we need to extract the data in a particular syntax. For example, we need to extract the lines containing following format –

```
X-DSPAM-Confidence: 0.8475
X-DSPAM-Probability: 0.0000
```

The line should start with `X-`, followed by 0 or more characters. Then, we need a colon and white-space. They are written as it is. Then there must be a number containing one or more digits with or without a decimal point. Note that, we want dot as a part of our pattern string, but not as meta character here. The pattern for regular expression would be –

```
^X-.*: [0-9.]+
```

The complete program is –

```
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if re.search('^X\S*: [0-9.]+', line):
          print(line)
```

The output lines will as below –

```
X-DSPAM-Confidence: 0.8475
X-DSPAM-Probability: 0.0000
X-DSPAM-Confidence: 0.6178
X-DSPAM-Probability: 0.0000
X-DSPAM-Confidence: 0.6961
X-DSPAM-Probability: 0.0000
……
……
```

Assume that, we want only the numbers (representing confidence, probability etc) in the above output. We can use *split()* function on extracted string. But, it is better to refine regular expression. To do so, we need the help of parentheses.

When we add parentheses to a regular expression, they are ignored when matching the string. But when we are using ***findall()***, parentheses indicate that while we want the whole expression to match, we only are interested in extracting a portion of the substring that matches the regular expression.

```
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    x = re.findall('^X-\S*: ([0-9.]+)', line)
    if len(x) > 0:
        print(x)
```

Because of the parentheses enclosing the pattern above, it will match the pattern starting with X- and extracts only digit portion. Now, the output would be –

```
['0.8475']
['0.0000']
['0.6178']
['0.0000']
['0.6961']
.....................
.....................
```

Another example of similar form: The file *mbox-short.txt* contains lines like –

```
Details: http://source.sakaiproject.org/viewsvn/?view=rev&rev=39772
```

We may be interested in extracting only the revision numbers mentioned at the end of these lines. Then, we can write the statement –

```
x = re.findall('^Details:.*rev=([0-9.]+)', line)
```

The regex here indicates that the line must start with `Details:`, and has something with `rev=` and then digits. As we want only those digits, we will put parenthesis for that portion of expression. Note that, the expression `[0-9]` is greedy, because, it can display very large number. It keeps grabbing digits until it finds any other character than the digit. The output of above regular expression is a set of revision numbers as given below –

```
['39772']
['39771']
['39770']
['39769']
.........................
.........................
```

Consider another example – we may be interested in knowing time of a day of each email. The file *mbox-short.txt* has lines like –

```
From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008
```

Here, we would like to extract only the hour 09. That is, we would like only two digits representing hour. Hence, we need to modify our expression as –

```
x = re.findall('^From .* ([0-9][0-9]):', line)
```

Here, `[0-9][0-9]` indicates that a digit should appear only two times. The alternative way of writing this would be -

```
x = re.findall('^From .* ([0-9]{2}):', line)
```

The number 2 within flower-brackets indicates that the preceding match should appear exactly two times. Hence `[0-9]{2}` indicates there can be exactly two digits. Now, the output would be –

```
['09']
['18']
['16']
['15']
.....
.....
```

3.4.4 Escape Character

As we have discussed till now, the character like dot, plus, question mark, asterisk, dollar etc. are meta characters in regular expressions. Sometimes, we need these characters themselves as a part of matching string. Then, we need to escape them using a back-slash. For example,

```
import re
x = 'We just received $10.00 for cookies.'
y = re.findall('\$[0-9.]+',x)
```

Output:

```
['$10.00']
```

Here, we want to extract only the price $10.00. As, $ symbol is a metacharacter, we need to use \ before it. So that, now $ is treated as a part of matching string, but not as metacharacter.

3.4.5 Bonus Section for Unix/Linux Users

Support for searching files using regular expressions was built into the Unix OS. There is a command-line program built into Unix called ***grep*** (Generalized Regular Expression Parser) that behaves similar to ***search()*** function.

```
$ grep '^From:'  mbox-short.txt
```

Output:

```
From: stephen.marquard@uct.ac.za
From: louis@media.berkeley.edu
From: zqian@umich.edu
From: rjlowe@iupui.edu
```

Note that, *grep* command does not support the non-blank character `\S`, hence we need to use `[^ ]` indicating not a white-space.