REGULAR EXPRESSIONS

A regular expression is a string that contains special symbols and characters to find and extract the information needed by us from the given data. A regular expression helps us to search information, match, find and split information as per our requirements. A regular expression is also called simply *regex*.

Regular expressions are used to perform the following important operations:

- Matching strings
- Searching for strings
- Finding all strings
- Splitting a string into pieces
- Replacing strings

The following methods belong to 're' module that are used in the regular expressions:

- match() method searches in the beginning of the string and if the matching string is found, it returns an object that contains the resultant string, otherwise it returns None. We can access the string from the returned object using group() method.

- search() method searches the string from beginning till the end and returns the first occurrence of the matching string, otherwise it returns None. We can use group() method to retrieve the string from the object returned by this method.

- findall() method searches the string from beginning till the end and returns all occurrences of the matching string in the form of a list object. If the matching strings are not found, then it returns an empty list. We can retrieve the resultant strings from the list using a for loop.

- split() method splits the string according to the regular expression and the resultant pieces are returned as a list. If there are no string pieces, then it returns an empty list. We can retrieve the resultant string pieces from the list using a for loop.

- sub() method substitutes (or replaces) new strings in the place of existing strings. After substitution, the main string is returned by this method.

Sequence characters in regular expressions

Sequence characters match only one character in the string. Let us list out the sequence characters which are used in regular expressions along with their meanings in Table 1.

Character	**Its description**
\d	Any digit (0 to 9)
\D	Any non-digit

\s	White space. Ex: \t\n\r\f\v
\S	Non-white space character
\w	Any alphanumeric (A to Z, a to z, 0 to 9)
\W	Non-alphanumeric
\b	A space around words
\A	Matches only at start of the string
\Z	Matches only at end of the string

Each of these sequence characters represents a single character matched in the string. For example '\w' indicates any one alphanumeric character. Suppose we write it as [\w]*. Here '*' represents 0 or more repetitions. Hence [\w]* represents 0 or more alphanumeric characters.

Example 1. A regular expression to search for strings starting with m and having total 3 characters using search() method.

```
import re
str = 'man sun mop run'
result = re.search(r'm\w\w', str)
if result:   # if result is not None
    print(result.group())

Output:
man
```

Example 2. A regular expression to search for strings starting with m and having total 3 characters using findall() method.

```
import re
str = 'man sun mop run'
result = re.findall(r'm\w\w', str)
print(result)

Output:
['man', 'mop']
```

Example 3. A regular expression using match() method to search for strings starting with m and having total 3 characters.

```
import re
str = 'man sun mop run'
result = re.match(r'm\w\w', str)
print(result.group())

Output:
man
```

Example 4. A regular expression using match() method to search for strings starting with m and having total 3 characters.

```
import re
str = 'sun man mop run'
result = re.match(r'm\w\w', str)
```

```
print(result)

Output:
None
```

Example 5. A regular expression to split a string into pieces where one or more numeric characters are found.

```
impor re
str = 'gopi 2222 vinay 9988 subba rao 89898'
res = re.split(r'\d+\b', str)
print(res)

Output:
['gopi ', ' vinay ', ' subba rao ', '']
```

Example 6. A regular expression to replace a string with a new string.

```
import re
str = 'Kumbhmela will be conducted at Ahmedabad in India.'
res = re.sub(r'Ahmedabad', 'Allahabad', str)
print(res)

Output:
Kumbhmela will be conducted at Allahabad in India.
```

Quantifiers in regular expressions

In regular expressions, some characters represent more than one character to be matched in the string. Such characters are called ‘quantifiers’

Character	Its description
+	1 or more repetitions of the preceding regexp
*	0 or more repetitions of the preceding regexp
?	0 or 1 repetitions of the preceding regexp
{m}	Exactly m occurrences
{m, n}	From m to n. m defaults to 0. n to infinity.

Example7. A regular expression to find all words starting with ‘an’ or ‘ak’.

```
import re
str = 'anil akhil anant arun arati arundhati abhijit ankur'
res = re.findall(r'a[nk][\w]*', str)
print(res)

Output:
['anil', 'akhil', 'anant', 'ankur']
```

Example 8. A regular expression to retrieve date of births from a string.

```
import re
str = 'Vijay 20 1-5-2001, Rohit 21 22-10-1990, Sita 22 15-09-2000'
res = re.findall(r'\d{2}-\d{2}-\d{4}', str)
```

```
print(res)

Output:
['22-10-1990', '15-09-2000']
```

NOTE: Try the following regex in the above example:

```
res = re.findall(r'\d{1,2}-\d{1,2}-\d{4}', str)
```

Special characters in regular expressions

Characters with special significance shown in the following Table can be used in regular expressions.

Character	Its description
\	Escape special character nature
.	Matches any character except new line
^	Matches beginning of a string
$	Matches ending of a string
[...]	Denotes a set of possible characters. Ex: [6b-d] matches any characters '6', 'b', 'c' or 'd'
[^...]	Matches every character except the ones inside brackets. Ex: [^a-c6] matches any character except 'a', 'b', 'c' or '6'
(...)	Matches the regular expression inside the parentheses and the result can be captured.
R | S	Matches either regex R or regex S

Example 9. A regular expression to search whether a given string is starting with 'He' or not.

```
import re
str = "Hello World"
res = re.search(r"^He", str)
if res:
    print ("String starts with 'He'")
else:
    print("String does not start with 'He'")

Output:
String starts with 'He'
```

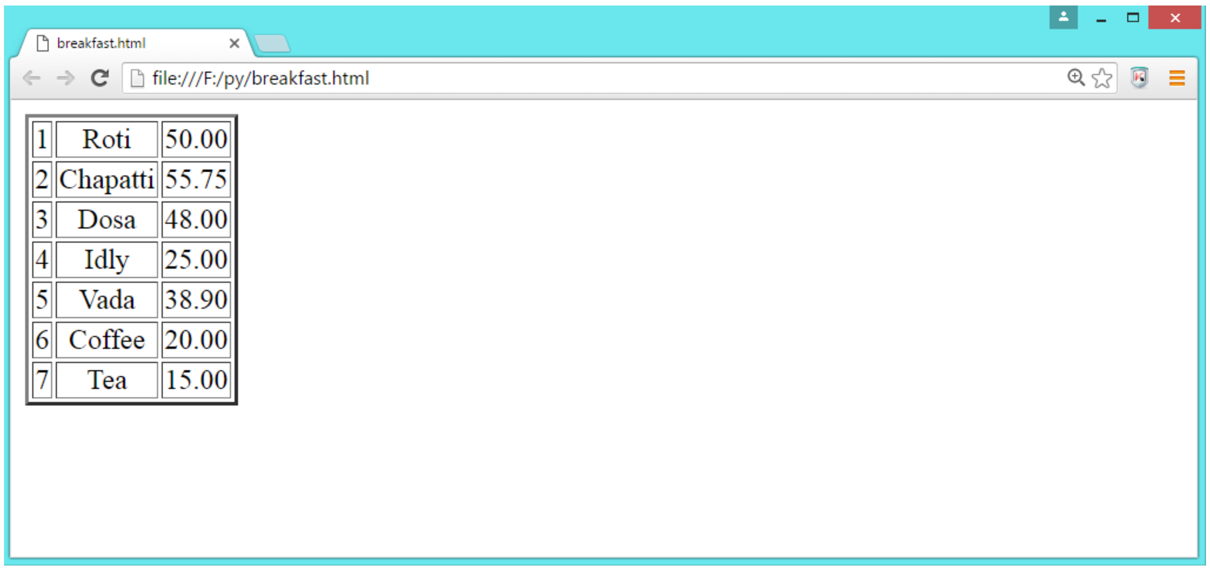
Retrieving information from a HTML file

Let us see how to apply regular expressions on a HTML file and retrieve the necessary information. As an example, let us take a HTML file that contains some items for breakfast and their prices in the form of a table, as shown here:

```
<! breakfast.html>
<html>
<table border=2>
<tr align="center"><td>1</td> <td>Roti</td> <td>50.00</td></tr>
<tr align="center"><td>2</td> <td>Chapatti</td> <td>55.75</td></tr>
<tr align="center"><td>3</td> <td>Dosa</td> <td>48.00</td></tr>
<tr align="center"><td>4</td> <td>Idly</td> <td>25.00</td></tr>
```

```
<tr align="center"><td>5</td> <td>Vada</td> <td>38.90</td></tr>
<tr align="center"><td>6</td> <td>Coffee</td> <td>20.00</td></tr>
<tr align="center"><td>7</td> <td>Tea</td> <td>15.00</td></tr>
</table>
</html>
```

When we open this file, we can see the table in the browser as shown below:



1	Roti	50.00
2	Chapatti	55.75
3	Dosa	48.00
4	Idly	25.00
5	Vada	38.90
6	Coffee	20.00
7	Tea	15.00

Let us assume that this file is available in our computer in the directory as: F:\py\breakfast.html. To open this file, we have to use urlopen() method of urllib.request module in Python. So, we have to use the following code:

```
import urllib.request
f = urllib.request.urlopen(r'file:///f|py\breakfast.html')
```

Observe the raw string passed to urlopen() method. It contains the path of the .html file, as:

```
file:///f|py\breakfast.html
```

The first word 'file:///' indicates file URL scheme that is used to refer to files in the local computer system. The next word 'f|py' indicates the drive name 'f' and the sub directory 'py'. In this, we have the file breakfast.html. Once this file is open, we can read the data using read() method, as:

```
text = f.read()
```

But the data in the HTML files would be stored in the form of byte strings. Hence, we have to decode them into normal strings using decode() method, as:

```
str = text.decode()
```

Now, we have the string ‘str’. We have to retrieve the required information from this string using a regular expression. Suppose we want to retrieve only item name and price, we can write:

r'<td>\w+</td>\s<td>(\w+)</td>\s<td>(\d\d.\d\d)</td>'

Please observe that the preceding expression contains three special characters: the first one is a \w+, the second one is (\w+) and the third one is (\d\d.\d\d). They are embedded in the tags <td> and </td>. So, the information which is in between the tags is searched.

The first \w+ indicates that we are searching for a word (item number). The next \w+ is written inside parentheses (). The parentheses represents that the result of the regular expression written inside these parentheses will be captured. So, (\w+) stores the words (item names) into a variable and the next (\d\d.\d\d) stores the words (item prices) into another variable. If we use findall() method to retrieve the information, it returns a list that contains these two variables as a tuple in every row. For example, the first two values are ‘Roti’ and ’50.00’ which are stored in the list as a tuple as: [('Roti', '50.00')] .

PROGRAMS

40. To retrieve item name and its price from a HTML file using a regular expression.

```
# web scraping using regex
import re
import urllib.request

# open the html file using urlopen() method
f = urllib.request.urlopen(r'file:///f|py\breakfast.html')

# read data from the file object into text string
text = f.read()

# convert the byte string into normal string
str = text.decode()

# apply regular expression on the string
# here /s is for space
result =
re.findall(r'<td>\w+</td>\s<td>(\w+)</td>\s<td>(\d\d.\d\d)</td>', str)

# display result
print(result)

# display the items of the result
for item, price in result:
   print('Item= %-15s Price= %-10s' %(item, price))

# close the file
f.close()
```