4. Python Regular Expression

4.1. Powerful pattern matching and searching

4.2. Power of pattern searching using regex in python

4.3. Real time parsing of data using regex

4.4. Password, email, URL validation using regular Expression

4.5. Pattern finding programs using regular expression

Regular Expressions in Python

The term Regular Expression is popularly shortened as **regex**. A regex is a sequence of characters that defines **a search pattern**, used mainly for performing **find and replace operations in search engines and text processors.**

Python offers regex capabilities through the “**re module”** bundled as a part of the standard library.

Raw strings

Different functions in Python’s **re module** use raw string as an argument. A normal string, when prefixed with **'r' or 'R'** becomes a raw string.

Example: Raw String

```
>>> rawstr = r'Hello! How are you?'
>>> print(rawstr)
Hello! How are you?
```

The difference between **a normal string** and a raw string is that the normal string in print() function translates escape characters (such as **\n**, **\t** etc.) if any, while those in a raw string are not.

Example: String vs Raw String

```
str1 = "Hello!\nHow are you?"
print("normal string:", str1)
str2 = r"Hello!\nHow are you?"
print("raw string:",str2)
```

Output

```
normal string: Hello!
How are you?
raw string: Hello!\nHow are you?
```

In the above example, \n inside str1 (normal string) has translated as a newline being printed in the next line. But, it is printed as \n in str2 - a raw string.

meta characters

Some characters carry a special meaning when they appear as a part pattern matching string. In Windows or Linux, DOS commands, we use ***** and **?** - they are similar to meta characters. **Python's re module** uses the following characters as meta characters:

. ^ $ * + ? [ ] \ | ( )

Metacharacters are characters with a special meaning:

Character	**Description**	**Example**
[ ]	**A set of characters**	**"[a-m]"**
\	**Signals a special sequence (can also be used to escape special characters)**	**"\d"**
.	**Any character (except newline character)**	**"he..o"**
^	**Starts with**	**"^hello"**
$	**Ends with**	**"world$"**
*****	**Zero or more occurrences**	**"aix*"**
+	**One or more occurrences**	**"aix+"**

{}	**Exactly the specified number of occurrences**	**"al{2}"**
|	**Either or**	**"falls|stays"**
( )	**Capture and group**	

When a set of alpha-numeric characters are placed inside square brackets [], the target string is matched with these characters. A range of characters or individual characters can be listed in the square bracket. For example:

Pattern	Description
[abc]	match any of the characters a, b, or c
[a-c]	which uses a range to express the same set of characters.
[a-z]	match only lowercase letters.
[0-9]	match only digits.

The following specific characters carry certain specific meaning.

Pattern	Description
\d	Matches any decimal digit; this is equivalent to the class [0-9].
\D	Matches any non-digit character
\s	Matches any whitespace character
\S	Matches any non-whitespace character
\w	Matches any alphanumeric character

Pattern	Description
\W	Matches any non-alphanumeric character.
.	Matches with any single character except newline '\n'.
?	**match 0 or 1 occurrence of the pattern to its left**
+	**1 or more occurrences of the pattern to its left**
*	**0 or more occurrences of the pattern to its left**
\b	boundary between word and non-word. /B is opposite of /b
[..]	Matches any single character in a square bracket
\	It is used for special meaning characters like . to match a period or + for plus sign.
{n,m}	Matches at least **n** and at most **m** occurrences of preceding
a| b	Matches either a or b

re.match() function

This function in **re module** tries to find if the specified pattern is present at the beginning of the given string.

re.match(pattern, string)

The function returns None, if the given pattern is not in the beginning, and a match objects if found.

Example: re.match()

```
 from re import match


mystr = "Welcome to TutorialsTeacher"
obj1 = match("We", mystr)
print(obj1)
obj2 = match("teacher", mystr)
print(obj2)
```

Output

```
<re.Match object; span=(0, 2), match='We'>
None
```

The match object has start and end properties.

Example:

```
>>> print("start:", obj.start(), "end:", obj.end())
```

Output

```
start: 0 end: 2
```

The following example demonstrates the use of the range of characters to find out if a string starts with 'W' and is followed by an alphabet.

Example: match()

```
from re import match

strings=["Welcome to TutorialsTeacher", "weather forecast","Winston Churchill",
"W.G.Grace","Wonders of India", "Water park"]

for string in strings:
    obj = match("W[a-z]", string)
    print(obj)
```

Output

```
<re.Match object; span=(0, 2), match='We'>
None
<re.Match object; span=(0, 2), match='Wi'>
None
<re.Match object; span=(0, 2), match='Wo'>
<re.Match object; span=(0, 2), match='Wa'>
```

re.search() function

The re.search() function searches for a specified pattern anywhere in the given string and stops the search on the first occurrence.

Example: re.search()

```
from re import search

string = "Try to earn while you learn"

obj = search("earn", string)
print(obj)
print(obj.start(), obj.end(), obj.group())
7 11 earn
Output
<re.Match object; span=(7, 11), match='earn'>
```

This function also returns the Match object with start and end attributes. It also gives a group of characters of which the pattern is a part of.

re.findall() Function

As against the search() function, the findall() continues to search for the pattern till the target string is exhausted. The object returns a list of all occurrences.

Example: re.findall()

```
from re import findall

string = "Try to earn while you learn"

obj = findall("earn", string)
print(obj)
Output
['earn', 'earn']
```

This function can be used to get the list of words in a sentence. We shall use \w* pattern for the purpose. We also check which of the words do not have any vowels in them.

Example: re.findall()

```
obj = findall(r"\w*", "Fly in the sky.")
print(obj)

for word in obj:
    obj= search(r"[aeiou]",word)
    if word!=' ' and obj==None:
        print(word)
```

Output

```
['Fly', '', 'in', '', 'the', '', 'sky', '', '']
Fly
sky
```

re.finditer() function

The re.finditer() function returns an iterator object of all matches in the target string. For each matched group, start and end positions can be obtained by span() attribute.

Example: re.finditer()

```
from re import finditer

string = "Try to earn while you learn"
it = finditer("earn", string)
for match in it:
    print(match.span())
```

Output

```
(7, 11)
(23, 27)
```

re.split() function

The re.split() function works similar to the split() method of **str object in Python**. It splits the given string every time when **a white space** is found. In the above example of the **findall()** to get all words, the list also contains each occurrence of **white space** as a word. **That is eliminated by the split() function in re module.**

Example: re.split()

```
from re import split

string = "Flat is better than nested. Sparse is better than dense."
words = split(r' ', string)
print(words)
```

Output

```
['Flat', 'is', 'better', 'than', 'nested.', 'Sparse', 'is', 'better', 'than', 'dense.']
```

re.compile() Function

The **re.compile()** function returns **a pattern object** which can be repeatedly used in different **regex functions**. In the following example, a string '**is**' is compiled to get a pattern object and is subjected to the **search()** method.

Example: re.compile()

```
from re import *

pattern = compile(r'[aeiou]')
string = "Flat is better than nested. Sparse is better than dense."
words = split(r' ', string)
for word in words:
    print(word, pattern.match(word))
```

Output

```
Flat None
is <re.Match object; span=(0, 1), match='i'>
```

```
better None
than None
nested. None
Sparse None
is <re.Match object; span=(0, 1), match='i'>
better None
than None
dense. None
```

The same pattern object can be reused in searching for words having vowels, as shown below.

Example: search()

```
for word in words:
    print(word, pattern.search(word))
```

Output

```
Flat <re.Match object; span=(2, 3), match='a'>
is <re.Match object; span=(0, 1), match='i'>
better <re.Match object; span=(1, 2), match='e'>
than <re.Match object; span=(2, 3), match='a'>
nested. <re.Match object; span=(1, 2), match='e'>
Sparse <re.Match object; span=(2, 3), match='a'>
is <re.Match object; span=(0, 1), match='i'>
better <re.Match object; span=(1, 2), match='e'>
than <re.Match object; span=(2, 3), match='a'>
dense. <re.Match object; span=(1, 2), match='e'>
```

The sub() Function

The sub() function replaces **the matches** with **the text of your choice:**

Example

Replace every **white-space character** with the number 9:

```
import re
```

```
txt = "The rain in Spain"
x = re.sub("\s", "9", txt)
print(x)
```

4.4. Password, email, URL validation using re

Password Validation using re

First we create a regular expression which can satisfy the conditions required to call it a valid password. Then we match the given password with the required condition using the search function of re. In the below example the complexity requirement is we need **at least one capital letter, one number and one special character.** We also need the length of the password to be between **8 and 18.**

```
import re

pswd = input("Enter a Valid password :  ")
reg = "^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*#?&])[A-Za-z\d@$!#%*?&]{8,18}$"

# compiling regex
match_re = re.compile(reg)

# searching regex
res = re.search(match_re, pswd)

# validating conditions
if res:
  print("Valid Password")
else:
  print("Invalid Password")
```

URL Validation using re

Following python code in which we pass the regex expression and string to search() method and checked whether the input URL matches the expression pattern. If the pattern is matched, the URL is valid else the URL is invalid.

```
import re


def check_url(ip_url):
    # Regular expression for URL
    regex = re.compile(
        r'^(?:http|ftp)s?://'    # http:// or https://
        r'(?:(?:[A-Z0-9](?:[A-Z0-9-]{0,61}[A-Z0-9])?\.)+(?:[A-Z]{2,6}\.?|[A-Z0-9-]{2,}\.?)|'
                        #domain...
        r'localhost|'    #localhost...
        r'\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})' # ...or ip
        r'(?::\d+)?      # optional port
        r'(?:/?|[/?]\S+)$', re.IGNORECASE)


    if (ip_url == None):
        print("Input string is empty !!!")
    if(re.search(regex, ip_url)):
        print("Input URL is valid !!!")
    else:
        print("Input URL is invalid !!!")
ch = 'y'
while ch == 'y':
    ip_url = input("Enter the URL string: ")
    check_url(ip_url)
    ch = input("Do you want to continue? (y or n): ")
    if ch == 'y':
        continue
    else:
        break
```

Output:

```
Enter the URL string: http://localhost:8080
Input URL is valid !!!
Do you want to continue? (y or n): y
Enter the URL string: http://172.16.16.16
Input URL is valid !!!
Do you want to continue? (y or n): n
```

Email Validation using re

Given a string, write a Python program to check if the string is a valid email address or not. An email is a string (a subset of ASCII characters) separated into two parts by @ symbol, a "personal_info" and a domain, that is personal_info@domain.

Valid Domain Name: A domain name consists of minimum two and maximum 63 characters. All letters from a to z, all numbers from 0 to 9 and a hyphen (-) are possible.

A domain name must not consist of a hyphen (-) on the third and fourth position at the same time.

Examples:

Input: ankitrai326@gmail.com

Output: Valid Email

Input: my.ownsite@ourearth.org

Output: Valid Email

Input: ankitrai326.com

Output: Invalid Email

In this program, we are using search() method of re module. So let's see the description about it.

re.search() : This method either returns None (if the pattern doesn't match), or re.MatchObject that contains information about the matching part of the string. This method stops after the first match, so this is best suited for testing a regular expression more than extracting data.

```
# Python program to validate an Email
# import re module
import re
# Make a regular expression for validating an Email.
regex = r"(^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$)"
 # Define a function for validating an Email.
 def check(email):
    if(re.search(regex, email)):
        print("Valid Email")
    else:
        print("Invalid Email")

if __name__ == '__main__':
    # Enter the email
    email = "ankitrai326@gmail.com"
    check(email)
   email = "my.ownsite@our-earth.org"
    check(email)
   email = "ankitrai326.com"
    check(email)
```

Output:

```
Valid Email
Valid Email
Invalid Email
```

```
List of Valid Email Addresses
email@example.com
firstname.lastname@example.com
email@subdomain.example.com
firstname+lastname@example.com
email@123.123.123.123
email@[123.123.123.123]
"email"@example.com
1234567890@example.com
email@example-one.com
_______@example.com
email@example.name
email@example.museum
email@example.co.jp
firstname-lastname@example.com
```

4.5. Pattern finding programs using regular expression